HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Mathematics and Systems Analysis

Juha Wiljakka

Reliability Analysis of Digital Transmission Software

This Master's Thesis has been submitted for official examination for the degree of Master of Science in Engineering in Espoo, Finland.

The Supervisor of the Thesis	Professor Raimo P. Hämäläinen
The Instructors of the Thesis	Dr. Tech. Urho Pulkkinen M.Sc. Martti Raivola

Espoo 5.11.1996

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Juha Wiljakka
Työn nimi:	Digitaalisen tiedonsiirto–ohjelmiston luotettavuusanalyysi
Päivämäärä:	5.11.1996 Sivumäärä: 64
Osasto:	Tietotekniikan osasto
Laitos:	Matematiikan ja Systeemianalyysin laitos
Professuuri:	Mat–2 Sovellettu matematiikka
Työn valvoja:	Prof. Raimo P. Hämäläinen
Työn ohjaajat:	TKT Urho Pulkkinen DI Martti Raivola

Ohjelmiston luotettavuus vaikuttaa merkittävällä tavalla ohjelmiston laatuun. Diplomityön tarkoituksena on tutkia ohjelmiston luotettavuusteorian ja –mallinnuksen soveltamista digitaalisen tiedonsiirto–ohjelmiston tuotekehitykseen.

Nykyaikainen digitaalinen tiedonsiirtolaitteisto pohjautuu monimutkaiseen ohjelmistorakenteeseen. Useita eri ohjelmistokerroksia tarvitaan alkaen HW–läheisestä tietoliikennesolmun ohjelmistosta päätyen PC– tai UNIX–pohjaiseen solmun– ja verkonhallintaohjelmistoon.

SDH–siirtojärjestelmälaitteiden systeemitestaus on vaativa tehtävä. Suuri määrä toiminnallisuuksia ja eri laitekombinaatioita on kyettävä testaamaan luotettavasti. Tärkeintä on luoda hyvä ohjelmiston testausympäristö, jossa mahdolliset ohjelmistoviat pystytään saamaan esiin mahdollisimman varhaisessa vaiheessa. On hyödyllistä vertailla peräkkäisiä ohjelmistoversioita. Se toteutetaan analysoimalla luotuja vikaraportteja.

Diplomityö on laadittu ohjelmiston luotettavuuden perusteokseksi Nokian SDH–osastolle. Työn tarkoituksena on kehittää vikaraportointia ja suunnitella viikottainen ohjelmiston luotettavuuden seurantaohjelma. Tämä mahdollistaa luotettavuusmallinnuksen käytön tulevaisuudessa. Perusteet SDH:sta, luotettavuusteoriasta, ohjelmiston testauksesta ja vikaraportoinnista kuuluvat olennaisena osana työhön. Lopuksi esitetään systeemitestaussuunnitelma seuraavalle Nokian SDH–ohjelmistoversiolle.

Avainsanat: SDH, synkroninen digitaalinen hierarkia, ohjelmiston luotettavuus, digitaalinen transmissio, ohjelmiston testaus

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Juha Wiljakka
Name of the thesis:	Reliability Analysis of Digital Transmission Software
Date:	5.11.1996 Number of pages: 64
Faculty:	Information Technology
Department:	Mathematics and Systems Analysis
Professorship:	Mat–2 Applied Mathematics
Supervisor:	Professor Raimo P. Hämäläinen
Instructors:	Dr.Tech. Urho Pulkkinen M.Sc. Martti Raivola

Software reliability is probably the most important part of the concept software quality. The thesis investigates the possibility to apply the software reliability theory and modelling to the product development of digital transmission software.

Modern digital transmission equipment is very much based on a complex software structure. Several layers of software are needed: from node software working closely with hardware to PC or UNIX based node/network management applications.

The system/integration test process of Synchronous Digital Hierarchy equipment is a challenging task. There is a large number of functionalities and combinations that have to be verified. The most important task is to create such an operational profile that will reveal the possible software faults. It is useful to make comparisons between previous releases. This is done by analyzing the failure reports created during software testing.

The thesis is going to be a software reliability handbook for SDH software development. One goal is to improve fault reporting and create a software reliability weekly follow–up system. This will make the use of reliability modelling possible. The principles of SDH, software reliability theory, software testing strategies and defect management are also introduced. Finally, a system test plan for the next SDH software release is introduced.

Keywords: SDH, Synchronous Digital Hierarchy, Software Reliability, Digital Transmission, Software Testing

FOREWORD

This master's thesis has been written in the SDH system testing department of Nokia Telecommunications.

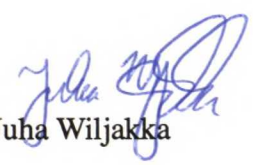
I want to thank professor Raimo P. Hämäläinen for his supervision.

I also want to thank my instructors Dr. Tech. Urho Pulkkinen and M.Sc. Martti Raivola for their valuable instructions when writing the thesis. I am specially grateful to my superior Martti Raivola for the best possible working atmosphere in the SDH system test laboratory. The colleagues at the SDH product development department were also of great help on encouraging and giving advice on the course of writing my thesis. Many thanks.

My parents, Anja and Heikki Wiljakka, I want to give my best thanks for their support during my studies. They always have understanding and encouraging attitude towards me. Without them, my graduation would not have been possible.

Last, but not least, I want to thank Satu for her optimism and great attitude towards my work.

Espoo, November 5th, 1996



Juha Wiljakka

TABLE OF CONTENTS

Diplomityön tiivistelmä
Abstract of the Master's Thesis
Foreword
Table of Contents
Abbreviations

1 Introduction **1**

2 Synchronous Digital Hierarchy **2**
2.1 Overview 2
2.2 Transmission Speed Rates 3
2.3 SDH Frame Structure 3
2.4 Multiplexing Signals to an SDH Signal 4
2.5 SDH Node Types 5
2.6 SDH Network Architecture 6

3 SYNFONET **8**
3.1 Overview 8
3.2 Synfonet STM 1/4 Node Equipment 9
3.3 Synfonet Node Manager 10
3.4 Network Management Applications 11
3.5 Future Releases 12
3.6 Overall System Software Architecture 13
3.7 Challenges in Synfonet Reliability 14

4 Product Process Milestones and Phases in Nokia Telecommunications **16**
4.1 Product Process in Nokia Telecommunications 16
4.2 Milestones 16
4.3 Phases 16
4.4 Testing Terminology 17

5 Defect Management Using Action Request System **20**
5.1 Overview 20
5.2 Description of ARS Applied to SDH Products Fault Reporting 20
5.3 Creating a Fault Report 21
5.3.1 Filling Different Fields in ARS 21
5.3.2 Sorting Found Faults according to Severity 22
5.4 Defect Lifecycle 24
5.5 Software Testing Cycle 25

6 Introduction to Software Reliability **26**
6.1 Overview 26
6.2 Software Reliability Models 29
6.2.1. Execution Time Component 30
6.2.2. Calendar Time Component 32
6.3 Estimation 34
6.4 Collecting Data to the Models 35

7 Reliability Analysis of Synfonet SDH Software **36**
7.1 Introduction 36
7.2 Reliability of the SDH Equipment 36
7.3 Testing in Practice 36
7.4 Failure Descriptions in Synfonet Products 37
7.4.1 Node Software Failure Types 37
7.4.2 Synfonet Node Manager Failure Types 38
7.5 Comparison between Releases 39
7.6 Failures in the Node Software 39
7.6.1 Failure Distributions 39
7.6.2 Cumulative Failures and Failure Frequencies 41
7.6.3 An Example of the Use of Reliability Models 47
7.6.4 Impact of the Code Size on the Number of Failures 49
7.7 Failures in Synfonet Node Manager 51
7.7.1 Failure Distributions 51
7.7.2 Cumulative Failures and Failure Frequencies 52
7.7.3 Impact of the Windows Platform 56
7.8 Statistical Analysis of Failure Distributions 57
7.9 Collecting Failure Data in Next Releases 59
7.10 System Test Plan for Synfonet C2.20 / C3.20 60

8 Conclusions **62**

References **63**

Appendix 1: An Example of Finding a Software Failure and Writing a Fault Report

Appendix 2: A Fault Report from VISE to ARS during Maintenance

Appendix 3: A News Release about an SDH contract

ABBREVIATIONS

ADM	Add–Drop Multiplexer
AIS	Alarm Indication Signal
ANSI	American National Standards Institute
AR	Action Request (a Fault Report made using ARS)
ARS	Action Request System (Trademark of Remedy Corporation)
ASIC	Application Specific Integrated Circuit
ASW	Application Software
ATM	Asynchronous Transfer Mode
AU	Administrative Unit
AUG	Administrative Unit Group
CCITT	Comité Consultatif International Télégraphique (present ITU–T)
CU	Control Unit
DCC	Data Communications Channel
DCCR	Data Communications Channel in RSOH
DCCM	Data Communications Channel in MSOH
DXC	Digital Cross Connect (node)
ETSI	European Telecommunications Standards Institute
FMEA	Failure Mode and Effects Analysis
GPF	General Protection Fault
HW	Hardware
IEEE	the Institute of Electrical and Electronics Engineers
ISDN	Integrated Services Digital Network
ISO	International Standardisation Organisation
ITU–T	International Telecommunication Union – Telecommunication Standardization Sector
LAN	Local Area Network
MSOH	Multiplex Section Overhead
NE	Network Element
NFL	Node Functionality Layer
NMS	Network Management System
NNI	Network Node Interface
NSAP	Network Service Access Point
NTC	Nokia Telecommunications
OS68	Operating System used by Synfonet Node SW
OSI	Open Systems Interconnection
PC	Personal Computer
PDH	Plesiochronous Digital Hierarchy
POH	Path Overhead

Q3	ITU–T management interface
REG	Regenerator
RSOH	Regenerator Section Overhead
SAM	Synfonet Alarm Manager
SAN	Synfonet Access Node
SDH	Synchronous Digital Hierarchy
SLαPβ	System Line α Proposal β (used e.g. in node software internal release labeling)
SNC	Sub–Network Connection
SNM	Synfonet Node Manager
SOH	Section Overhead
SONET	Synchronous Optical Network (ANSI / North America)
SSW	System Switch (Unit)
STM	Synchronous Transport Module
SU	Service Unit
SW	Software
TCP–IP	Transmission Control Protocol – Internet Protocol
THD	Transmission Hardware Drivers
TM	Terminal Multiplexer
TMN	Telecommunications Management Network
TMS	Transmission Management System
TU	Tributary Unit
UP	Unit Protection
VC	Virtual Container
VISE	VIka SEuranta (fault reporting system, used during maintenance)
WWW	World Wide Web
λ(t)	Failure intensity function dμ/dt
μ(t)	(Expected) number of failures experienced by time t
μP	Microprocessor

1 INTRODUCTION

Sophisticated telecommunications systems are needed to satisfy the growing needs of the modern society. The telecommunications industry has become one of the major global industries during the last fifteen years and the growth is still very remarkable. Modern digital transmission equipment is based on optical transmission, integrated electronics and a complex software structure. Conventional telephone traffic is still very important but many other services, such as video signal, WWW and mobile phone communications need more and more transmission capacity.

There are many limitations in the widely used Plesiochronous Digital Hierarchy (PDH) transmission system. These are, for example, insufficient capacity for network management, lack of great bit rates and the use of different hierarchies around the world. Due to these limitations, planning started in the late 1980s to create a new standard – Synchronous Digital Hierarchy (SDH). SDH will also act as a basic transmission network for future ATM services including the Broadband–ISDN. The role of the software is extremely important in SDH transmission equipment.

Software reliability is probably the most important of the characteristics included in the concept software quality. Software reliability concerns itself with how well the software functions to meet the requirements of the customer. It can be simply defined as the probability that the software will work without failure for a specified period of time under specified conditions.

The master's thesis investigates the possibility of applying software reliability modelling in SDH software product development. The basics of Synchronous Digital Hierarchy, software reliability, product process, software testing and fault reporting are naturally presented. The thesis is prepared to be a software reliability handbook for Nokia SDH software development.

One goal of the thesis is to complete fault reporting instructions. The aim is the ability to create as informative fault reports as possible. A list of possible failure types in Synfonet, i.e. a negative specification of the system, is also created. An efficient software reliability weekly follow–up system is planned.

Chapter 2 is an introduction to the SDH world. There is discussion about SDH frame structure, node types and network architecture. Chapter 3 is a description about Nokia's Synfonet SDH equipment. Chapter 4 is about product process milestones and phases in Nokia Telecommunications including testing terminology and definitions. Defect management, i.e. fault reporting using Action Request System, is introduced in chapter 5. Instructions for creating high quality fault reports are given.

The theory of software reliability is introduced in chapter 6. A reliability analysis of Synfonet SDH Software using the Action Request System failure statistics is done in chapter 7. A plan for monitoring software reliability more efficiently in next releases is introduced. It includes the idea of creating weekly follow–up reports. The goal is to use the failure statistics more efficiently and apply reliability modelling in the future. Also, a system test plan proposal for the next Synfonet release is introduced.

2 SYNCHRONOUS DIGITAL HIERARCHY

2.1 Overview

Telecommunications operators are demanding higher quality transmission, more bandwidth and more flexibility in the network. The widely used transmission system Plesiochronous Digital Hierarchy (PDH) is not capable to fulfil all those demands.

In the mid 1980s standards for a System called SONET (Synchronous Optical NETwork) were developed in the USA. Since then, ITU–T (previously CCITT) has adapted the SONET standards as the Synchronous Digital Hierarchy (SDH). The basic idea of the SDH is to create a flexible, high capacity and reliable digital network in which the transmission is based on fibre optics.

In SDH, plesiochronous signals such as 64 kbit/s, 2 Mbit/s, 34 Mbit/s and 140 Mbit/s are containerized, or placed directly into the 155 Mbit/s SDH standard signal block Synchronous Transport Module 1 (STM–1). Higher transmission speed rates are STM–4, STM–16 and in the future STM–64.

SDH does not have the limitations of PDH. The most important drawbacks of PDH are the inability to identify individual channels in a higher order bit stream, insufficient capacity for network management, lack of greater bit rates than 140 Mbit/s and the use of different hierarchies around the world.

ITU–T recommendations G.707 (transmission rates), G.708 (network node interface), and G.709 (multplexing techniques) define the basic properties of the SDH. These three recommendations are unified to the new G7.07 that is called Network Node Interface for the Synchronous Digital Hierarchy [1]. There are also some later recommendations, such as G.781 – G.784 (multiplexing equipment), G.957 – G.958 (optical transmission systems), and G.826 (performance monitoring).

At the moment SONET is widely used in the USA and SDH is making its breakthrough in Europe and in Far East (e.g. Thailand and China). SDH will act as a bearer for future Asynchronous Transfer Mode (ATM) services including the Broadband–ISDN.

2.2 Transmission Speed Rates

ITU–T Recommendation G.707 [1] defines the transmission rates of SDH networks. The basic bit rate is 155.52 Mbit/s (STM–1) and the higher rates are exact multiples of the basic bit rate. The bit rates are presented in Table 1.

Table 1. *SDH transmission speed rates [1].*

Hierarchy level	Bit rate (Mbit/s)
STM–1	155.52
STM–4	622.08
STM–16	2488.32
STM–64	9953.28

2.3 SDH Frame Structure

ITU–T Recommendation G.707 is the Network Node Interface (NNI) for the SDH. It defines the structure of the Synchronous Transport Module (STM–N), the formats for mapping and multiplexing PDH and ATM elements into an STM–N frame, and the functionalities of the overhead bytes. This definition is to make sure that the transmission equipment manufactured by different manufacturers are compatible with each other.

STM–1 is the basic transport module in SDH. The STM–1 frame consists of 270 columns and nine rows (see Figure 1). It consists of payload data and transport overheads. Because one STM–1 frame is 2430 bytes, frame length is 125 µs and there are eight bits per byte, the STM–1 transmission speed rate is 155.52 Mbit/s. The higher capacity structures are generated by interleaving on a byte–by–byte basis N (N=4, 16, 64) of these basic STMs.

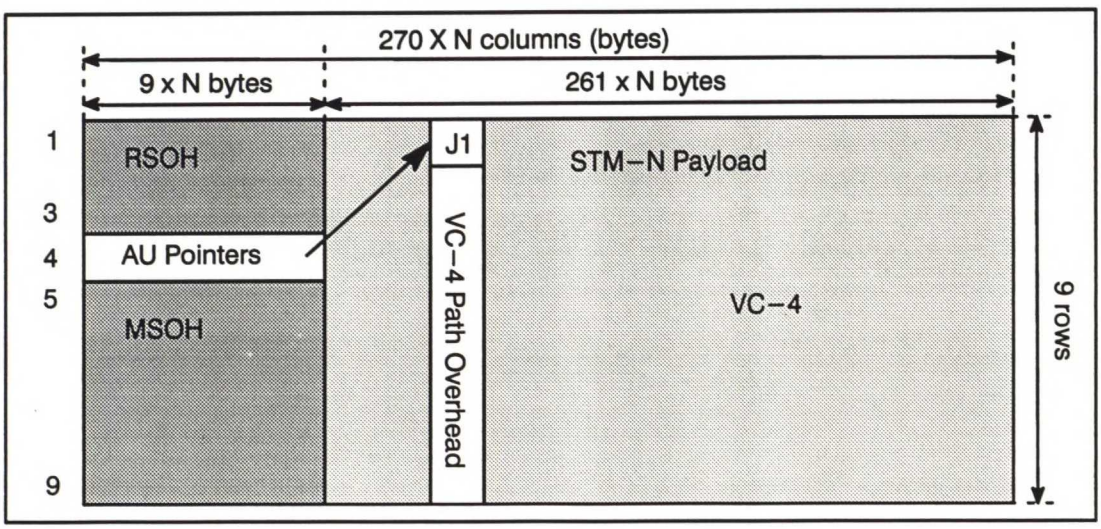


Figure 1. *The STM–N frame.*

The actual payload signaling rate in STM–1 signal is 149.76 Mbit/s. The payload is in Virtual Containers (VC–N), which include the Path Overhead (POH). The fourth row of Section Overhead (SOH) contains the AU–Pointers (AU=Administrative Unit). AU–Pointers point to the location of the highest VC within the STM–N frame.

SOH is divided into RSOH (Regenerator Section Overhead) and MSOH (Multiplex Section Overhead). An SDH regenerator node interpretes only the information of the RSOH. In an SDH multiplexer node the entire SOH information is interpreted.

Many important bytes used e.g. in network management and synchronization are located in RSOH and MSOH. E.g. bytes D1 – D3 in RSOH and bytes D4 – D12 in MSOH are so called data communication channels (total capacity 768 kbit/s). They are used as network management channels.

2.4 Multiplexing Signals to an SDH Signal

Figure 2 shows the multiplexing structure from PDH to SDH [1, p.6]. The term multiplexing is used when multiple lower path layer signals are adapted into a higher order path signal or when the higher order path signals are adapted into a multiplex section. Mapping is a process when tributaries are adapted into Virtual Containers (VC). Aligning happens e.g. between VC–12 and TU–12 when a pointer is included in TU–12 to be able to locate the first byte of VC–12.

As an example, 63 2.048 Mbit/s (in short format 2M) signals can be put in one STM–1 signal. One bytesynchronous 2M signal can contain 32 64 kbit/s signals (30 phone lines). So one STM–1 signal can carry 1890 phone calls and one STM–16 signal can carry 30240 phone calls.

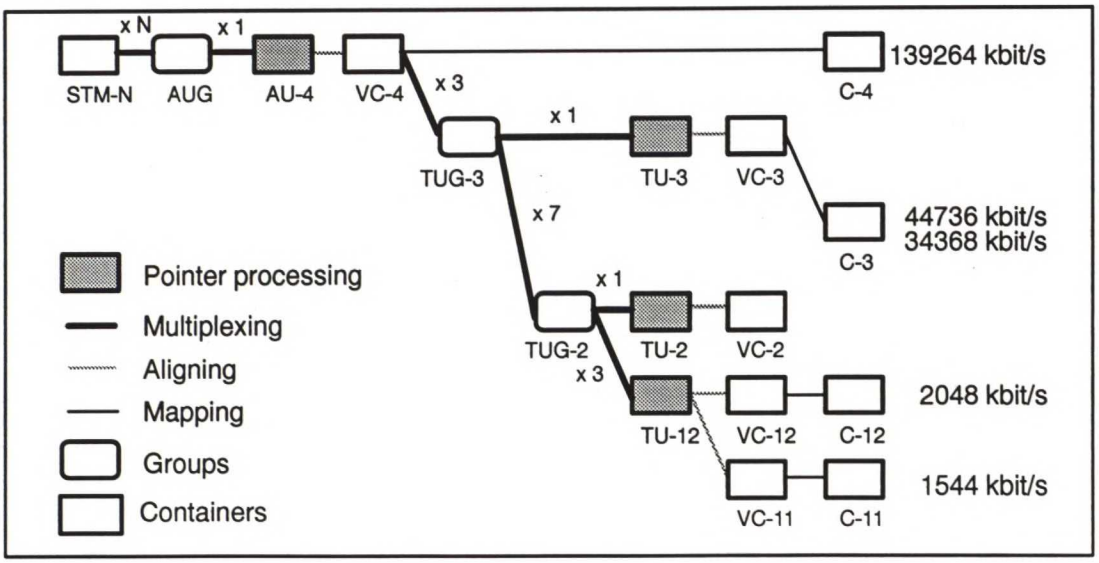


Figure 2. *ETSI SDH multiplexing structure [1].*

2.5 SDH Node Types

There are four basic SDH node types: regenerator (REG), terminal multiplexer (TM), add/drop multiplexer (ADM) and digital cross connect (DXC) nodes.

The Regenerator

Regenerators are used in long distance connections to amplify signals and to observe possible fault situations. Regenerators use the information of the RSOH bytes. There are different level STM regenerators.

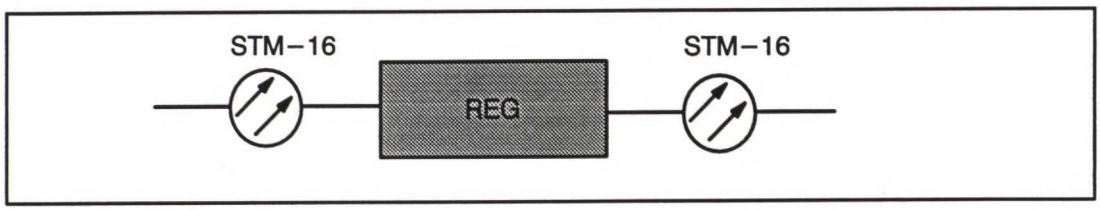


Figure 3. *An STM–16 Regenerator.*

The Terminal Multiplexer

Terminal Multiplexer combines different level PDH or SDH signals into one STM signal, e.g. STM–4. There are different types of TMs: STM–1, STM–4 and STM–16 TMs.

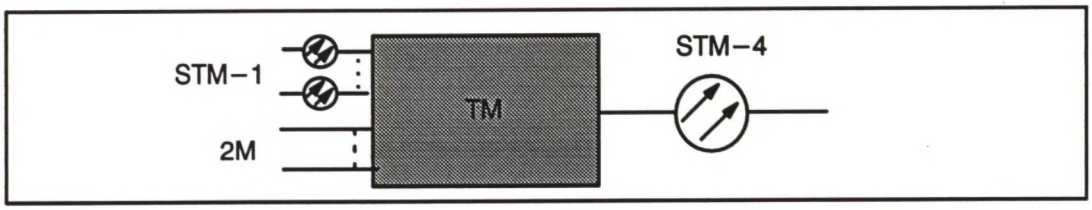


Figure 4. *An STM–4 Terminal Multiplexer.*

The Add/Drop Multiplexer

An STM–4 Add/Drop Multiplexer makes it possible to add e.g. some 2M PDH signals and STM–1 signals to STM–4 signals and also drop some 2M or STM–1 signals out of STM–4 signals. The ADM functionality is implemented making different level cross connections. There are also STM–1 and STM–16 ADMs.

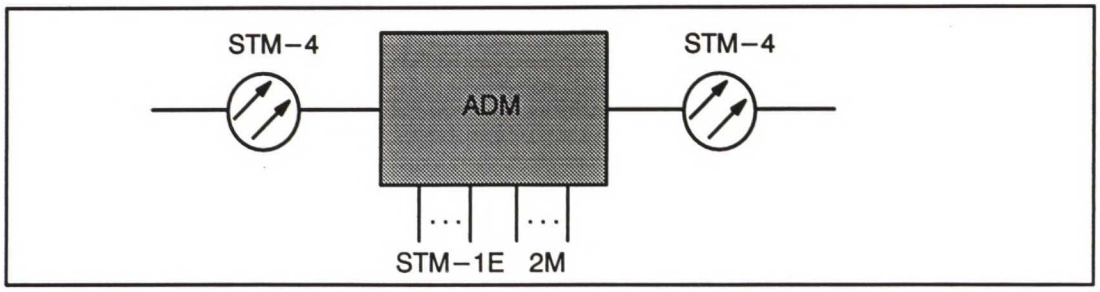


Figure 5. *An STM–4 Add/Drop Multiplexer.*

The Digital Cross Connect Node

The Digital Cross Connect node (DXC) is the most complex SDH node type. It typically has many different STM and PDH interfaces. The basic idea of DXC is to create different level cross connections (VC–12, VC–2, VC–3, VC–4) between different signals. Transmission paths can be protected using e.g. two STM–1 links (Sub–Network Connection (SNC) protection).

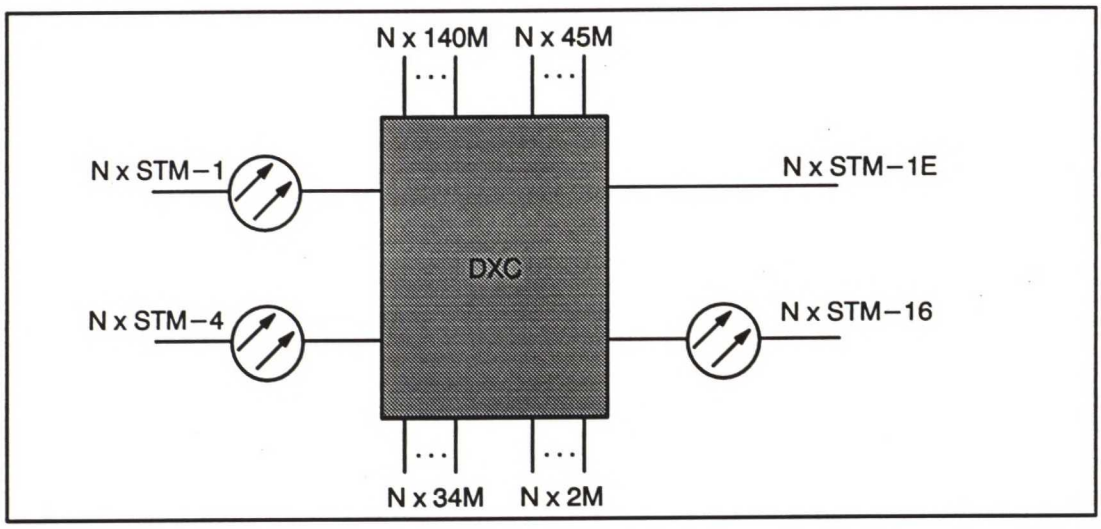


Figure 6. *A Digital Cross Connect Node.*

2.6 SDH Network Architecture

Four main network topologies used are point–to–point, chain, ring and mesh network. There is terminal multiplexer at each end of the point–to–point network. PDH networks typically are point–to–point networks. In a chain network ADM nodes are present to provide the ability to insert and remove traffic from the main transmission path. Ring networks are typical in SDH networks, there can be STM–1, STM–4 or STM–16 rings (see Figure 7). The protection situation is better compared to point–to–point or chain networks because there is always two alternative paths from node A to node B. In a mesh network elements are linked by STM links directly. See Figure 7 upper part to see an example of a mesh network.

SDH network architecture can be composed to four parts (see Figure 7).

- Basic Access
- Secondary Access
- Primary Access
- Long Distance or Regional Transit

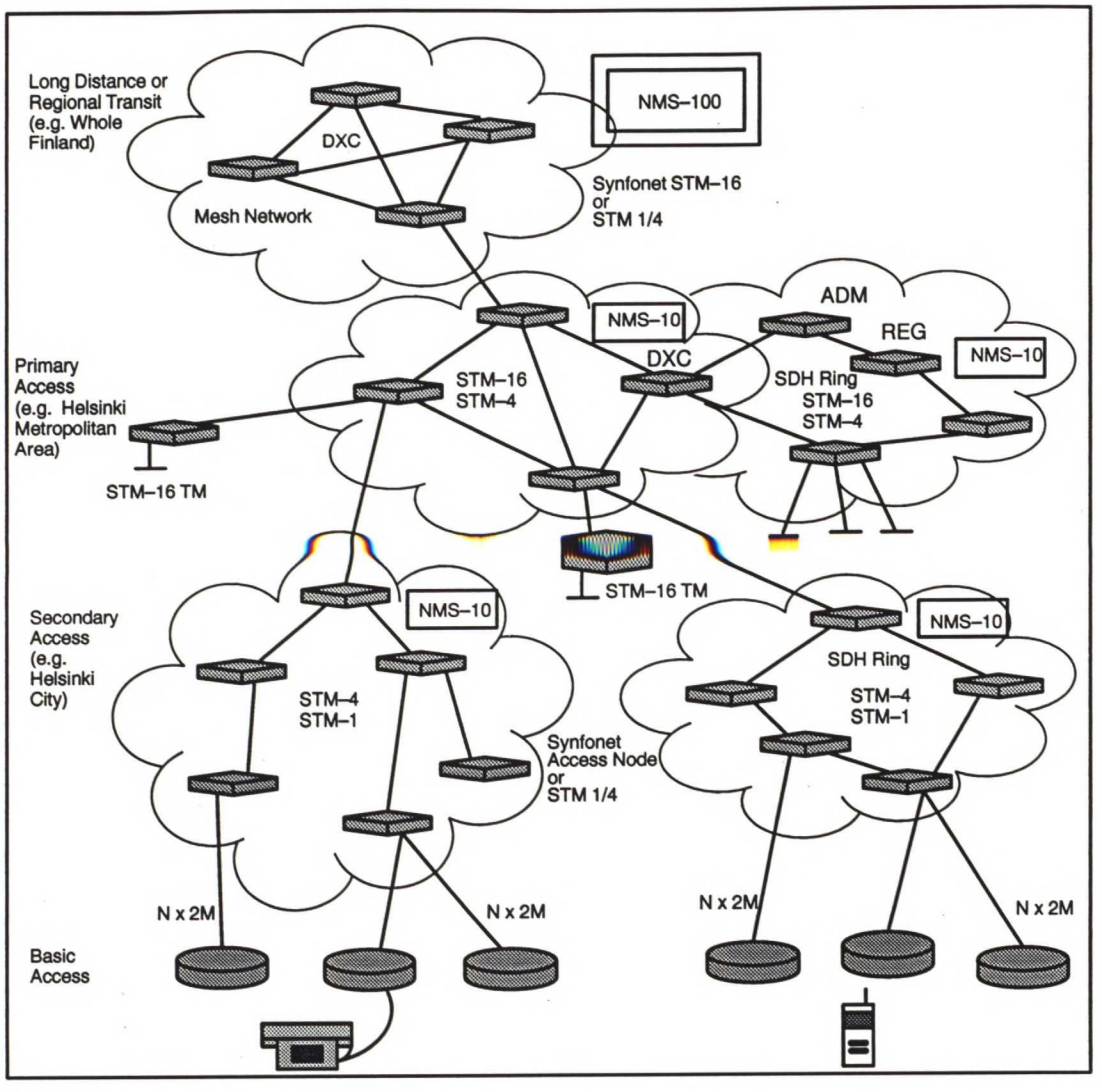


***Figure 7.** SDH network architecture.*

SDH provides a unified telecommunication network infrastructure, which can be characterized by a layered model. Long distance or regional transit layer is used for transporting inter regional or international traffic. These networks are typically meshed. Primary access networks transport information between large urban and metropolitan areas of a country. Primary access networks are usually STM–16 or STM–4 rings or meshed networks. Smaller urban and rural areas are covered by secondary access networks, which are often STM–4 or STM–1 rings or trees. Basic access layer is used for collecting traffic from access networks. Figure 7 depicts the way it is possible to utilize SYNFONET Product Family in each SDH network layer.

3 SYNFONET

3.1 Overview

Synfonet is Nokia's brand name for the family of SDH transmission equipment including node management applications. A Synfonet node is a basic network element. The name SYNFONET stands for SYNchronous Fibre Optical NETwork.

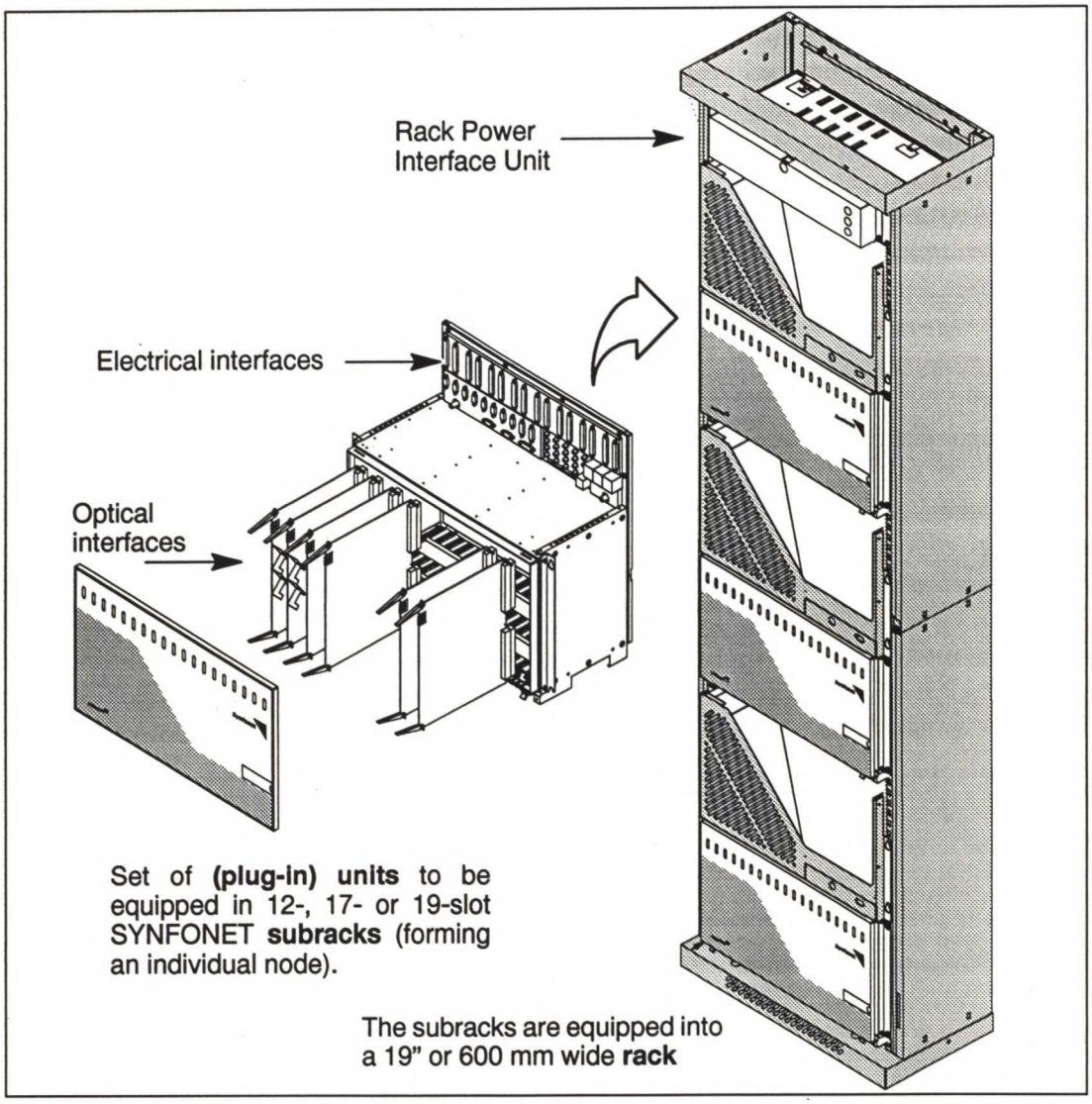


Figure 8. *Physical structure of Synfonet node equipment [2].*

3.2 Synfonet STM 1/4 Node Equipment

A Synfonet node is a flexible and variable network element that can be configured as a regenerator, terminal multiplexer, add/drop multiplexer or digital cross connect node. It consists of a subrack and a set of plug–in units for the electrical and optical interfaces and for the service, control and cross–connect functions. Same plug–in units can be used to configure different node types. There are three types of subracks: 12–, 17–, and 19–slot. Three subracks can be equipped into a 600 mm wide ETSI rack.

Nodes can be equipped with different hardware and functionality options. It is easy to tailor the equipment to the required applications. Synfonet nodes can later be easily upgraded. For example, an STM–1 ring network can be upgraded to STM–4 ring by upgrading the network nodes from STM–1 ADMs to STM–4 ADMs. Node upgrades are implemented by adding, removing or replacing plug–in units and then changing the software set–up using Windows based Synfonet Node Manager. The software in the plug–in units can be changed to an improved release offering more functionality. It is done by changing the memory circuits containing the unit programs in Synfonet C2.x releases. In Synfonet C3.x releases the new node software is downloaded to the plug–in units using TCP–IP protocol. After downloading the new software is activated.

The connector panel in the node has a BNC connector through which the node is connected to the Q3 management network (Ethernet). Usually the Ethernet cable is between a node and a PC in which the node management software is installed. The node and network management messages between nodes go using Data Communications Channels in STM–N Frame: DCCM in MSOH and DCCR in RSOH.

There are many types of plug–in units in Synfonet. The units used in Synfonet C2.1 / C3.0 equipment are CU (Control Unit), SSW (System Switch Unit), SU (Service Unit), STM–1, STM–4, TSW1 (Time Switch 1), 2M(TA) and STM1E / 140M. CU controls the node behaviour and takes care of the timing (synchronization) and the Q3 management traffic. SSW is responsible for (VC–12 / VC–4) cross connections. SU takes care of the auxiliary connections.

3.3 Synfonet Node Manager

The Synfonet Node Manager (SNM) is Windows based application for configuring, controlling and monitoring individual Synfonet nodes. SNM can manage geographically remote nodes because Ethernet connection is needed to only one node in the network. Other nodes are managed using Data Communications Channels.

SNM runs on PC environment under Microsoft Windows 3.1 or 3.11. Future releases are going to run under Windows 95 and Windows NT. These are the main functions of SNM:

- Installing / configuring new nodes
- Changing the configuration of previously configured nodes
- Monitoring the fault status (alarms) and performance of any node in the network

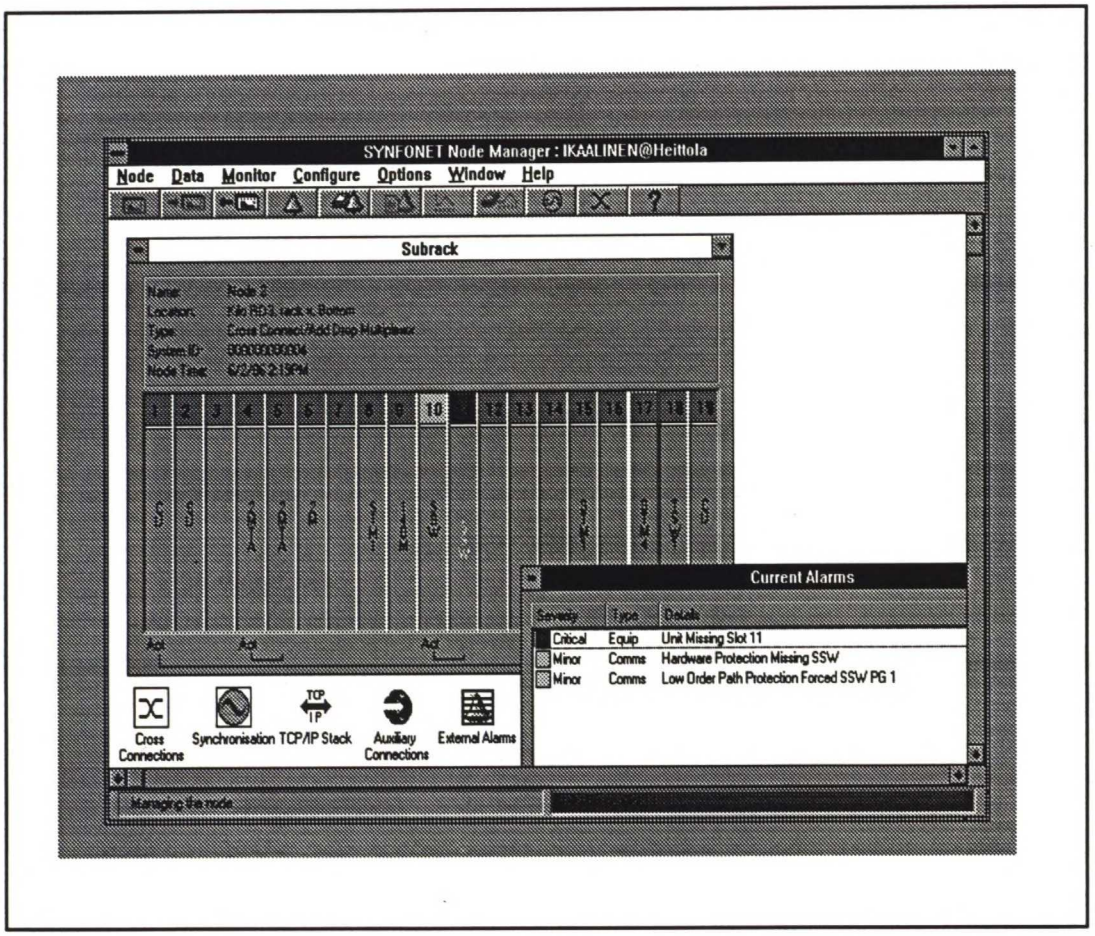


Figure 9. *Synfonet Node Manager user interface.*

3.4 Network Management Applications

Nokia Network Management System (NMS–X) product family consists of a range of network management applicatons. Here, two of them used in SDH networks are introduced.

NMS/10

Nokia Network Management System 10 (NMS/10) is element management system for transmission networks. The application can be used both in SDH and in PDH network management. The Synfonet Alarm Manager (SAM) running under NMS/10 is for SDH network management. NMS/10 runs under Microsoft Windows 3.1 or 3.11 and in the future Windows 95 / Windows NT.

The main functions of NMS/10 are:

- Obtaining and displaying alarm information from Synfonet nodes. Up to 50 nodes can be monitored simultaneously.
- Creating graphical network maps to show the real–time alarm status of Synfonet nodes operating in a network.
- Running the Synfonet Node Manager applications of the required release to modify the node settings in the network.

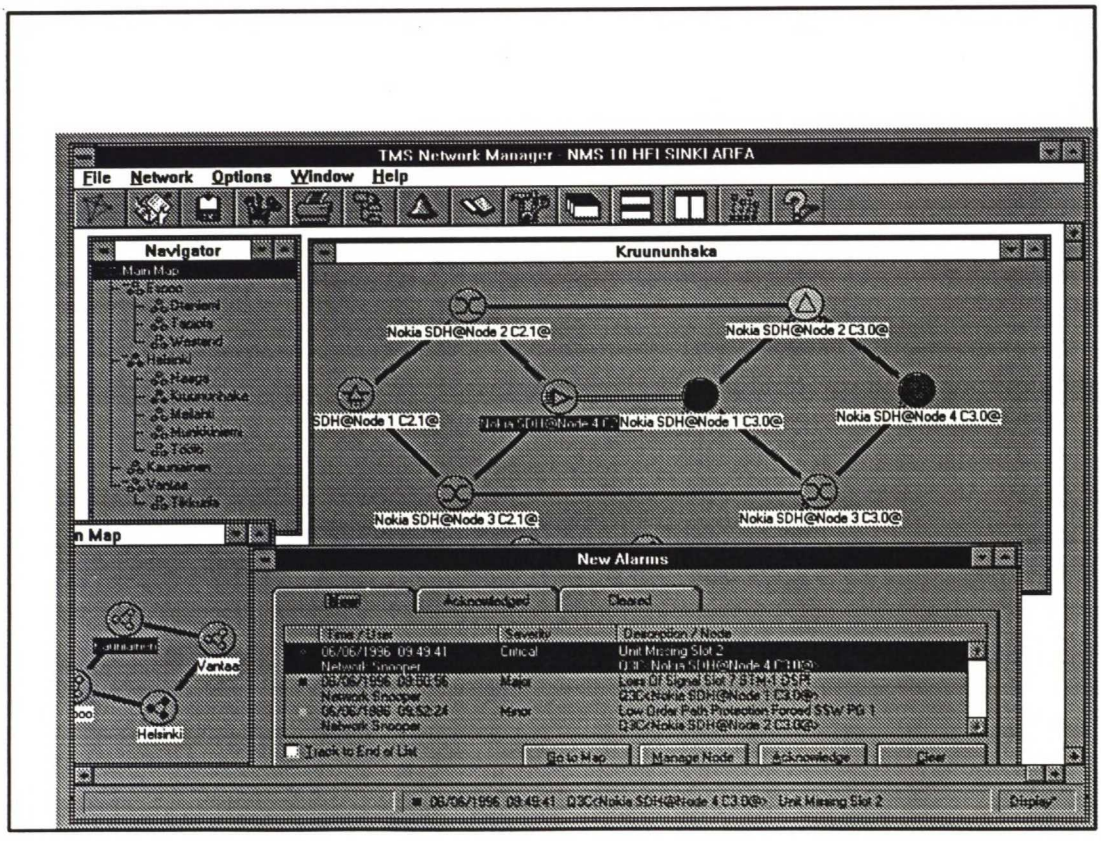


Figure 10. *NMS/10 (TMS–NetMan) application*

NMS/100

NMS/100 is a complete management system for big transmission networks. It runs in a UNIX workstation and has remarkably higher system requirements than NMS/10. It provides the following functions:

- Fault management
- Configuration management
- Performance management
- Security Management
- System administration
- New releases / functionality (software download)

3.5 Future Releases

Two new programs are running in Nokia SDH product development. These are called Synfonet Access Node and Synfonet STM–16.

Synfonet Access Node

Synfonet Access Node (SAN) is a future SDH transmission equipment manufactured by Nokia Telecommunications. It is going to be used in secondary access and basic access networks. SAN will be a cost–effective SDH solution down to the customer premises. The SDH interfaces in SAN will be STM–1 and STM–4. 2M and 34M PDH interfaces will be supported. Synfonet Access Node can be configured as a terminal multiplexer or an add/drop multiplexer.

Synfonet STM–16

Synfonet STM–16 is a future SDH transmission equipment to be used in regional transit (long distance) and primary access networks. Synfonet STM–16 is a same kind of concept as Synfonet STM 1/4. It is flexible and different node types can be configured using the same plug–in units and the same subrack. Synfonet STM–16 is designed to accommodate trunk, regional and high speed access networks (ATM including the broadband–ISDN) and it therefore supports STM–16, STM–4 and STM–1 level SDH signals. PDH interfaces are not supported. Synfonet STM–16 can be configured as a digital cross connect, an add/drop multiplexer, a terminal multiplexer or a regenerator node.

3.6 Overall System Software Architecture

The node software in the Synfonet node is divided in different functional parts. The main parts of the node software are Application Software (ASW), Q3 Stack, Node Functionality Layer (NFL), Transmission Hardware Drivers (THD) and Support SW [3].

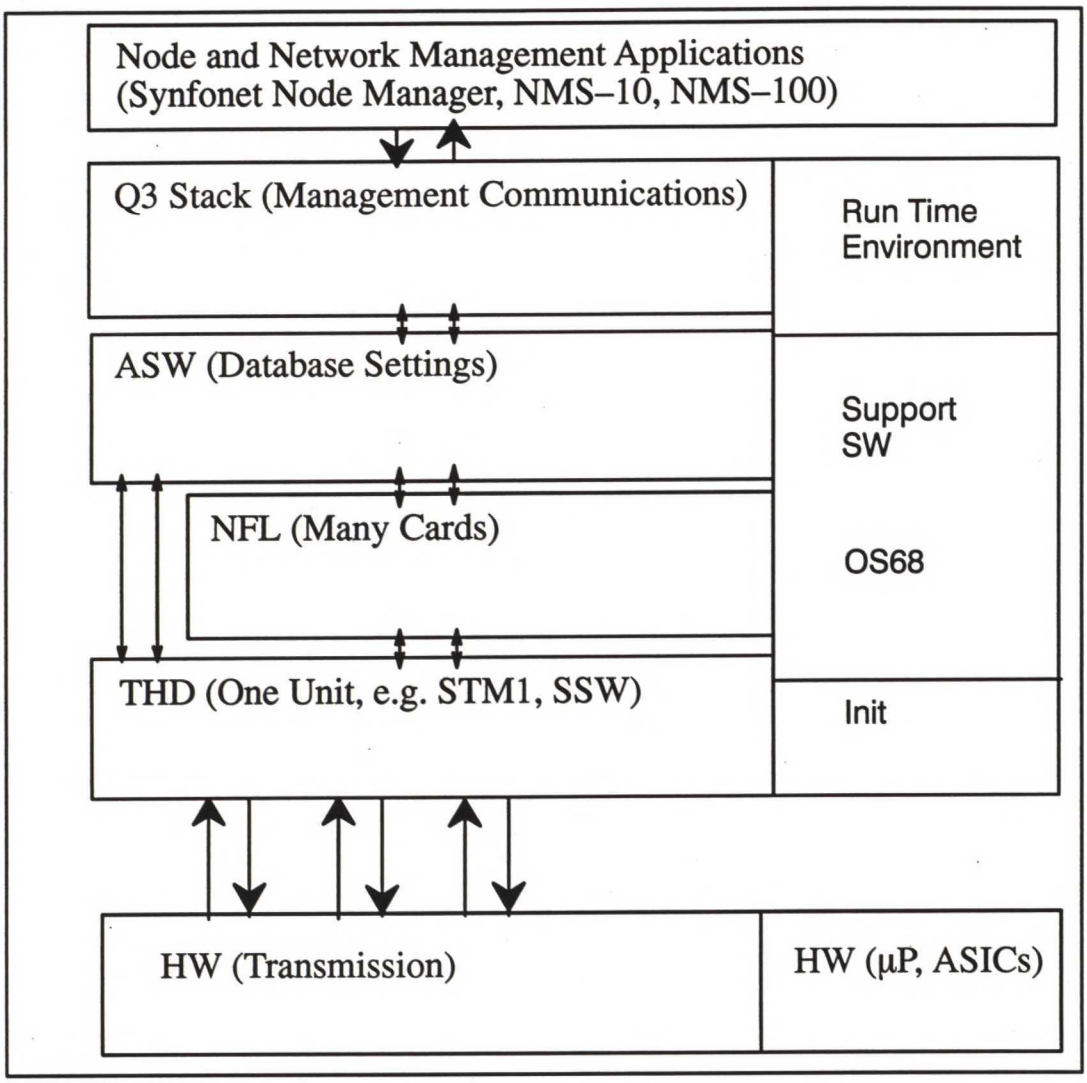


Figure 11. *Synfonet overall system software architecture [3].*

ASW

In OSI network management there are application processes called "managers" on management systems. ASW is the layer providing the application processes for the Synfonet network element. ASW components are called Handlers. ASW can be seen as a software layer between "real" functional node software and management applications such as Synfonet Node Manager.

Q3 Stack

Synfonet node communicates with the manager application (e.g. SNM) via the OSI protocol Q3 stack. The Q3 protocol stack ensures the reliability of message transfer by flow control.

NFL

The Node Functionality Layer (NFL) consists of components that control the functionality that needs co–operation of more than one card. NFL satisfies the communication needs between THDs on separate cards. With NFL components it is possible to simplify the interface to THD which otherwise would mean big amount of work for ASW.

THD

The Transmission Hardware Drivers (THD) are closest software components to hardware. They are components controlling the part of the transmission related hardware of one card that is defined to be under the responsibility of the specific THD. The main task of THD is to convert the HW interface into general form so, that the details of HW will be hidden from other software.

Support SW

The Support SW is a collection of components for hiding unnecessary distribution, OS68 and physical HW details from other SW. Support SW includes e.g. memory and backup management, software download and card configuration.

3.7 Challenges in Synfonet Reliability

Flexibility is one of the main features of Synfonet SDH equipment. It is also a great challenge to software/hardware development and testing. Using the same hardware and software it is possible to install a big number of different node types / combinations. The basic node types in Synfonet future release C2.20 / C3.20 are DXC, ADM, STM1 TM, STM4 TM, STM1 REG, and STM4 REG.

The software structure in Synfonet C2.20 is the same than in Synfonet C3.20. Only difference is the different compilations needed for different node plug–in unit hardwares. The situation is the same if we compare the current Synfonet C2.1 and C3.0 node software. C2.x releases use HW1 (manually changeable memory circuits for node software) and C3.x releases use HW2 (downloadable software). Naturally both versions C2.x and C3.x have to be tested.

If the number of different node types is analyzed, the hardware/software differences must be taken into account. In Synfonet C2.20 / C3.20 there are six different node types, three different subrack widths (19, 17 and 12 slot) and two different software versions (C2.20 and C3.20). Thus we get altogether 36 different nodes that should be verified! Of course, all these nodes share common software components and not all functionality has to be tested in each node. Every one of these 36 nodes can be configured in a different way (how many STM–N cards, 2M cards, 34M cards and is the HW unit protection used, etc.).

This is also a challenge to the installation procedure: every one of these nodes has to be installed using the same node manager (SNM). The role of the software is extremely important. Node software must be flexible to be installed in many different functional configurations.

Hardware reliability must not be forgotten. The reliability of hardware components such as node subracks, plug–in units, ASICs and lasers have a very important role in the system reliability. Hardware is usually thoroughly tested before the software testing starts. One role of the system test team is to make sure that the hardware and the software work perfectly together.

4 PRODUCT PROCESS MILESTONES AND PHASES IN NOKIA TELECOMMUNICATIONS

4.1 Product Process in Nokia Telecommunications

The Product Process Milestones have been harmonized in Nokia Telecommunications. All product lines, platform units as well as business units are using the same phased product process including eight decision milestones and seven phases between them [4].

The names of the milestones give a common "language" to different product lines when discussing about the status of different on–going product programs. Closer definition of each milestone is prepared by each product line / business unit.

4.2 Milestones

Milestone can be defined as a decision point for business decisions [4]. There are altogether eight milestones: E–1, E0 ... E5, E5+. The names and the basic definitions of the milestones are in Figure 12. The program management team shall present to the decision maker all necessary information needed for the specified decisions. It is good to remember that the major part of the decisions are made by program management team during the phases. When a milestone is reached, the product program (as an example Synfonet STM–16 program) is in certain state and it can be compared to some other program (e.g. Synfonet Access Node program).

When we talk about Research and Development system / integration testing, the most important time is between milestones E2 and E4 (see Figure 12). Most of the integration testing is done before milestone E3. System Test (including system verification and system validation) is done during phase 4 between milestones E3 and E4. In practise, it is not very clear where the limit of system and integration testing goes but this area is under study.

4.3 Phases

Phase can be defined as a period when certain program tasks are performed [4]. In each phase one part of the tasks is to prepare information, analysis, review reports etc. for milestone decisions. When defined tasks in phase N are performed, milestone N is reached.

The products for the first customer delivery will be manufactured and tested at the end of phase 4. The decision of the first customer delivery is made at milestone E4 (delivery start).

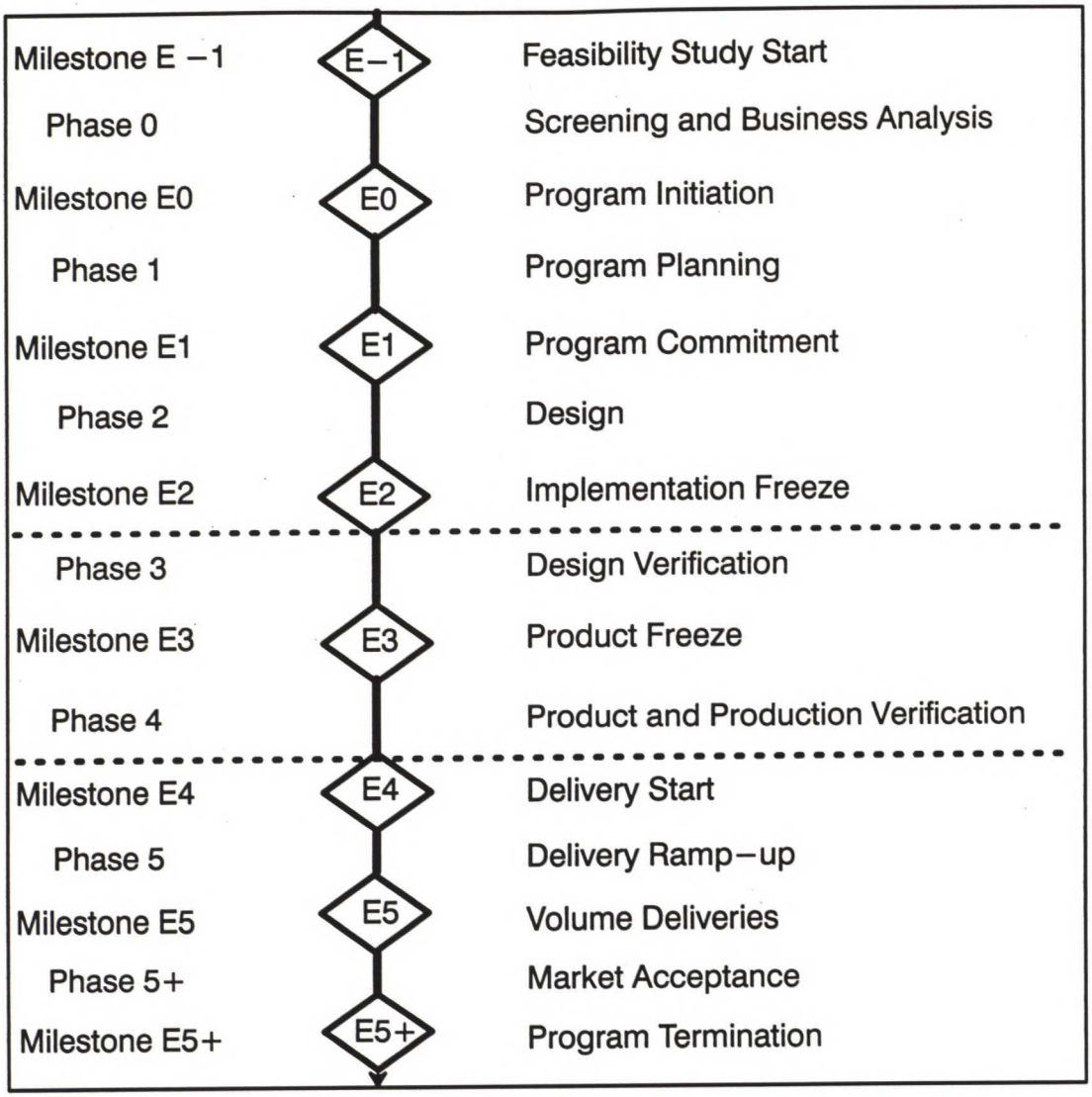


***Figure 12.** Product process milestones and phases in NTC.*

4.4 Testing terminology

There are many types of testing. Different testing types can be defined this way (adapted from [5]):

- Component testing (module testing) is done for individual hardware or software components or groups of related components.
- Integration testing is testing in which hardware components, software components or both are combined and tested to evaluate the interaction between them. It is verified that each module works correctly with the system. Integration testing is done on many different levels.
- System testing is done on a complete, integrated system to evaluate the system's compliance with its requirement specifications. System testing is made from the user's point of view.

- Verification and validation (used together as one term) is used to determine whether the requirements for the system are complete and correct, the products of each development phase fulfil the requirements or conditions imposed by the previous phase, and the final system or component complies with the specified requirements.
- Verification is done after each phase of development. Verification is a process of ensuring correctness and consistency of the system. Verification can be informal and doesn't require approved test specifications. If the product doesn't pass the verification phase, verification must be repeated after fixes are made to the product.
- Validation is the process of evaluating the system during or at the end of the development process to determine whether it satisfies the specified requirements. Validation follows successful verification and requires approved test specifications.

The structure of a software application can be defined as "the proper development of application systems to optimize technology and satisfy requirements". Some structural testing subcategories [6] are listed in Table 2.

Table 2. *Structural testing terms [6].*

Structural Testing Terms	***Description***
Stress and Volume Testing	Test processing of peak and steady loads of high volume data
Usability Testing	Evaluate human factor or usability problems
Compatibility / Conversion Testing	Verify compatibility with specified external data-bases, equipments, etc.
Performance Testing	Verify program meets specific performance efficiency objectives
Regression Testing	Determine whether any faults have been introduced while fixing another fault

Selecting test cases in the best possible way is very important. A good test case has a high probability to reveal an error, is repeatable, is not redundant with other test cases, and invokes as many input conditions as possible. The testing types can be divided in three groups: black–box testing, white–box (glass–box) testing, and grey–box testing. In black–box testing, the test cases are generated from specification and documentation and no deeper understanding of the code structure is needed. In white–box testing, test cases are generated from the source code using the information of the code structure. Grey–box testing is black–box oriented but utilizes knowledge of implementation. In most cases, SDH software testing is black–box testing.

The placement of different testing types in the NTC milestone model is described in Figure 13. Integration testing should be done by milestone E3. System tests including system verification and validation are to be done during phase 4.

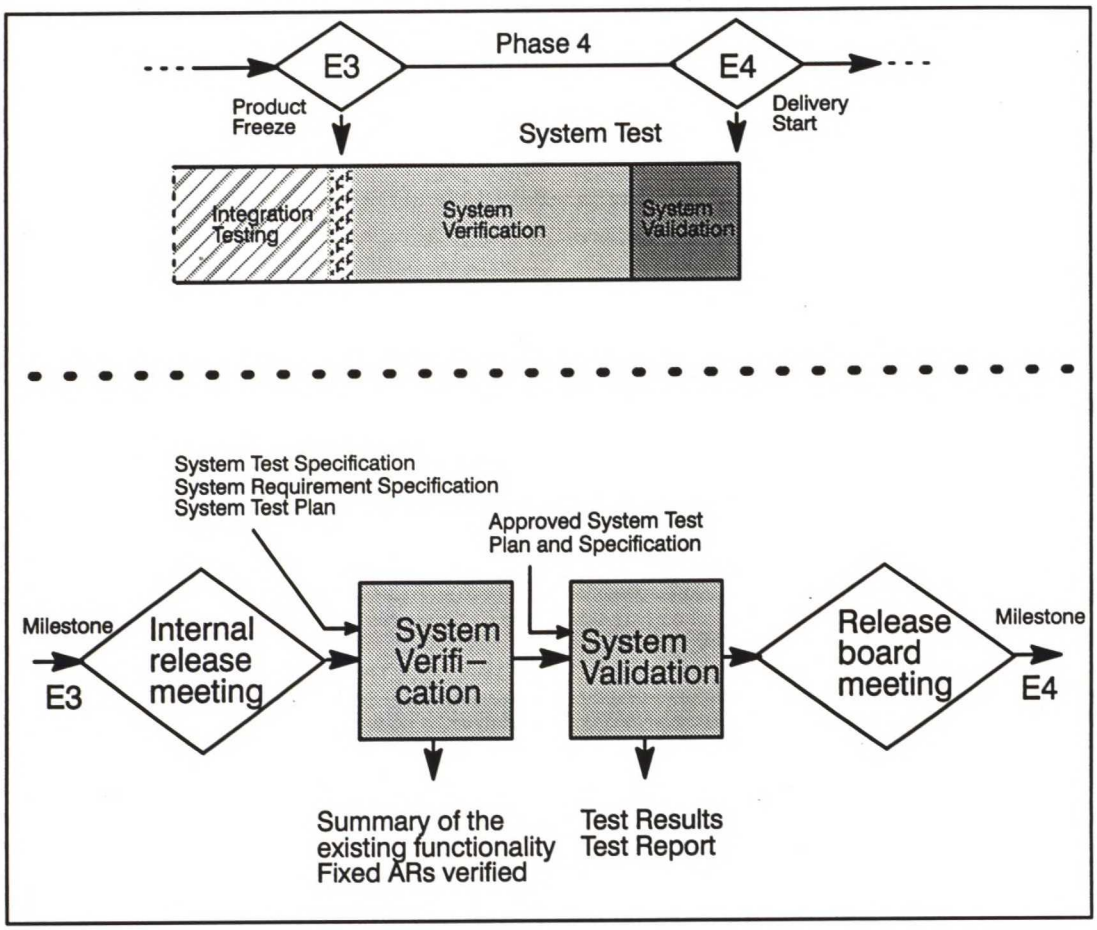


Figure 13. *System test process within a product process [5].*

5 DEFECT MANAGEMENT USING ACTION REQUEST SYSTEM

5.1 Overview

Action Request System (ARS) is a tool used for defect management in Nokia Telecommunications. Action Request System is a trademark of Remedy Corporation. The Action Request System tool used in NTC is customized internally by Nokia. ARS is used in SDH Product Development fault reporting. There is another tool for customer fault reporting called VISE (VIka SEuranta). The faults reported by customers are transferred manually into ARS.

5.2 Description of ARS Applied to SDH Products Fault Reporting

Action Request System runs on UNIX platform but it can also be used in a PC. ARS user interface is presented in Figure 14. The basic idea is that there are different fields on the screen. The writer of a fault report puts information in those fields. He both types in information and selects from lists. It is possible to do efficient queries from Action Request database using key words.

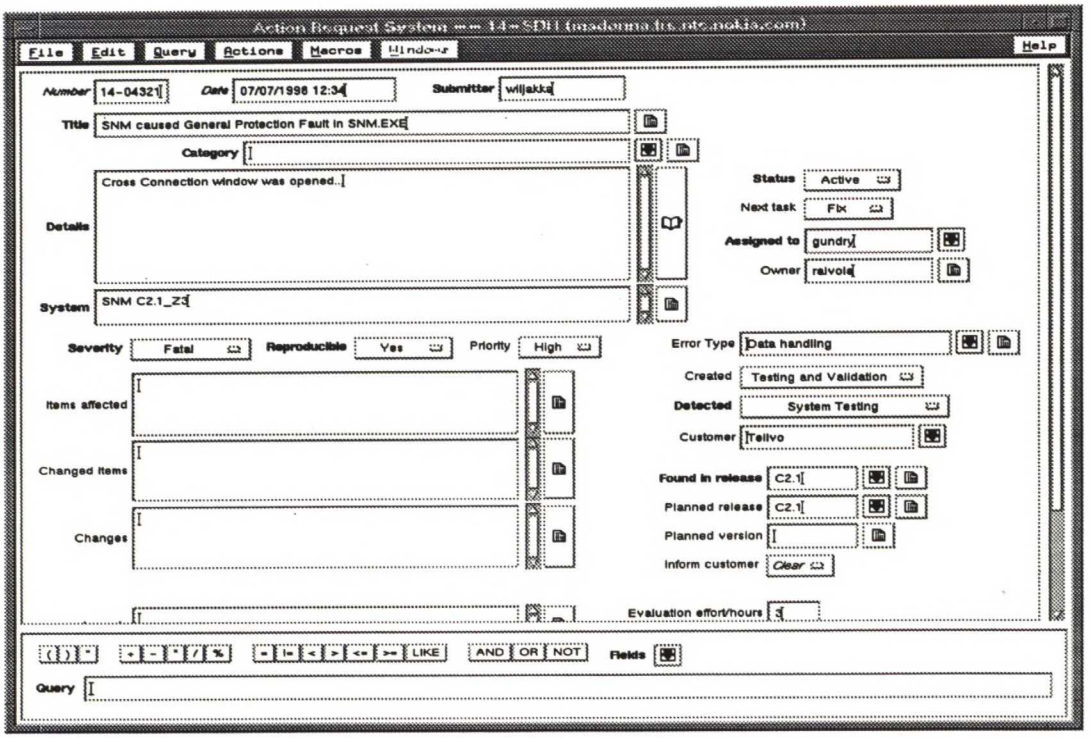


Figure 14. Action Request System user interface.

Action Request System is an on–line system. Created fault reports and changes in fault reports are updated immediately in AR database. User can make different kind of queries using key words and/or logical operators (AND, OR, etc.). User can also create reports to a file or print them on paper (e.g. report of faults found during last two weeks sorted by severity).

5.3 Creating a Fault Report

It is very important to create informative high quality fault reports. This section gives instructions on creating a good fault report. It is important to fill all fields and write down all the details in the fault report.

5.3.1 Filling Different Fields in ARS

These fields are the most important fields in ARS. The system requires filling them when submitting a new AR.

- *Submitter*
  The name of the AR submitter should be put in this field. There is a link to submitters E–mail address in the AR database.
- *Title*
  Title is a brief description of the problem. For example "SNM caused General Protection Fault in Module X when Cross Connections window was opened."
- *Category*
  Category is selected from the list. Main level categories are for example Synfonet SNM, Synfonet Node SW and Synfonet Node HW. ARS Defect Manager is responsible for updating the contents of the lists.
- *Details*
  As much information as possible should be written into this field. If the fault is reproducible, a pattern to reproduce it should be written. The details can be described like this:
  - Step-by-step details of how the problem was caused. Test case numbers for tests that reproduce the problem should be given, if possible.
  - Details about whether or not anyone else was using the system, what other applications were running, whether events were arriving from the manager etc.
  - References to any logs (e.g. some card terminal printouts), if they are available and seem relevant to the problem, are also written here. All logs that have been stored should be placed in a directory accessible to all, and the name of that directory (and file name) should be specified here.

  If there is some other information about the problem available, it should be mentioned here, e.g. the name of the person that has extra information for this problem.

- *System*
  The description of the system configuration, including both hardware and software system lines in use when the problem was observed is written here. For example "Synfonet Node SW C2.1 SL7P8, SNM C2.1 version B9 (build 020396), NMS/10 C3.0 build N2 and SAM C2.0 version 33."
- *Severity*
  See Section 5.3.2. for further details
- *Reproducible*
  Gives some indication of whether this problem is reproducible or not. The basic alternatives are 'Yes', 'No' and 'Sometimes'.
- *Detected*
  This indicates the area in which the (test) person was working when he / she detected the problem, e.g. 'System Test'.
- *Found in release*
  The release in which this problem was found should be put here.

The system updates next fields automatically: 'Number', 'Date' and 'Last modified by' (and the date of last modification). There are also some optional fields that may be filled in.

5.3.2 Sorting Found Faults according to Severity

In Action Request System every reported defect is sorted by severity. The severity of the fault priorizes the faults. Usually fatal faults are corrected faster than cosmetic ones. The severities are in Table 3 and there is also some sorting criteria. The criteria in [7] have been edited and completed.

Table 3. *Failure severity definitions.*

Menu Choice	Description
Fatal	The product/system is not usable with this problem; it crashes, severe transmission failures occur, or the product is data–corrupting. If this problem persists, the product can be useless. Good examples of fatal faults are Node SW crashes and SNM reproducible GPFs.
Major	The product/system can be used, but the problem causes a reduction in functionality. The product does not crash under normal operating conditions, but there are some features which cannot be used. Product/system defects that are Major have a high end–user impact.
Minor	Either there is an acceptable workaround, so the user can achieve the same objective differently, or there is no loss in functionality of the product. Product/system defects that are Minor have a low end–user impact.

Menu Choice	Description
Cosmetic	These are situations where the product does not appear as it should, for example the size or the text of a window is wrong. No functionality is affected in any way.
Enhancement	The product does exactly what it should, but a way to improve it has been identified. A good idea to improve the product is a typical enhancement.
Not Evaluated	The severity of the defect has not been evaluated yet. This shall not be the final severity value for the entry.
No Action	The AR requires no action. This can be because of one of 3 reasons: •The AR is a duplicate of a previously entered AR; •The AR has been evaluated as NOT representing a defect; •The AR describes a defect in some function that is no longer present. Reasoning behind this decision should be given in the Details field.

5.4 Defect Lifecycle

The defect lifecycle begins when the submitter (usually system test engineer) sends the Action Request to an engineering manager in the software project. Then the engineering manager, responsible for that part of software, becomes the owner of the AR. The owner assigns the AR to the evaluator to evaluate the defect (which part of the code needs fixing, when is the fixing to be done, is the AR a duplicate of previously found failure). It is usual that the owner evaluates the defect himself. Then the fixer is selected. Fixer is usually the SW engineer that has the best knowledge of that specific part of the code. The status of the AR is changed to 'Fixed' and the AR is sent to the owner, or sometimes even directly to the verifier. It is highly recommended that the verifier is the same person as submitter. The submitter usually has the best knowledge how to create the failure situation and how to reproduce it. Verifier verifies the fix in the next internal release of the program and accepts the fix or otherwise returns the AR back to the fixer. If the fix is accepted by the verifier, the AR is sent to the owner that closes it. In practice, the verifier can close it himself if the engineering manager has given a permission. Figure 15 represents the defect lifecycle phases.

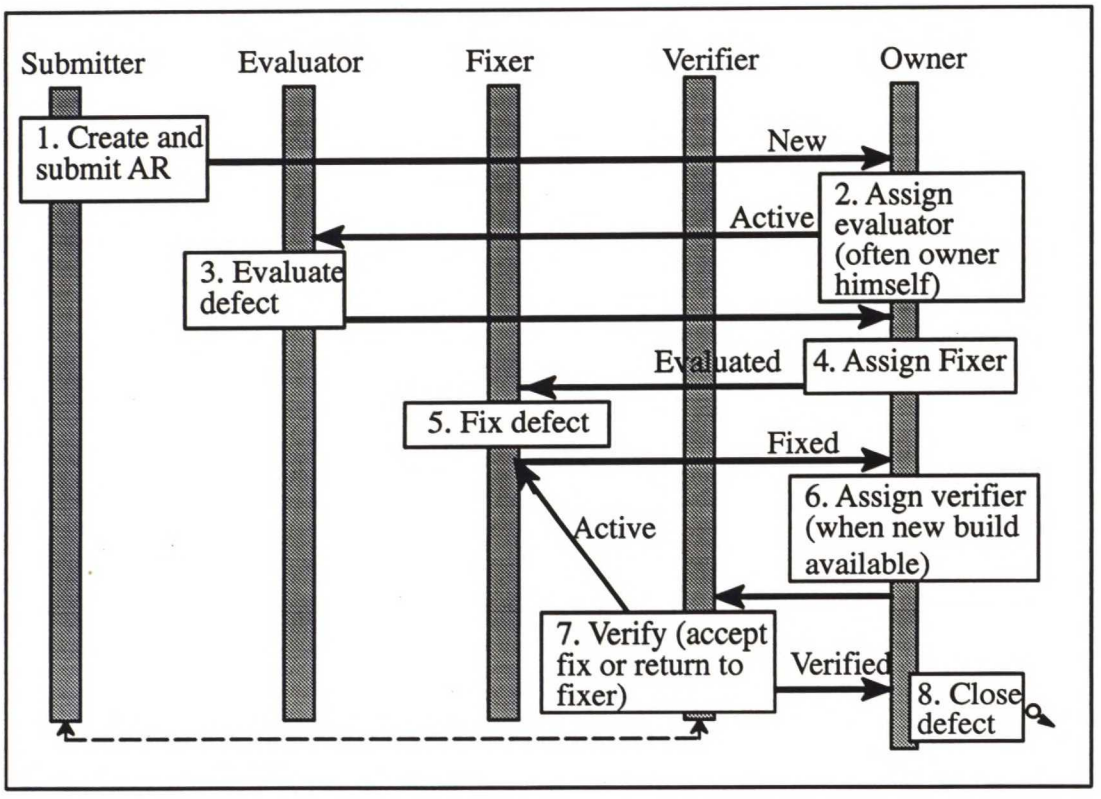


Figure 15. *Defect lifecycle phases, edited from [7].*

It is important to minimize the lifecycle of a defect. Thus the person the AR is assigned to, should react as soon as possible. Owner should evaluate the defect as soon as possible, and verifier should verify the fix as soon as the corrected version is released.

5.5 Software Testing Cycle

Figure 16 represents a testing cycle that is typical e.g. in Synfonet node software testing. Software department gives an internal release to be tested in the system test department. After that, there is a short delay before the software can be tested. The software must be installed in the nodes: using software downloading in Synfonet C3.x nodes, and burning node software to memory circuits in Synfonet C2.x nodes. Software downloading usually takes from one to three hours and burning node software to memory circuits can take even eight hours per node. The situation is same for Synfonet Node Manager. First, the old version has to be uninstalled in the PC and the new version has to be installed from the PC network.

After installing the new release, the testing begins. New faults are found and fault reports (ARs) are created. Also, the old fixed faults are verified and AR database is thus updated. Simultaneously, the software department is fixing faults and getting ready for the next internal release (build).

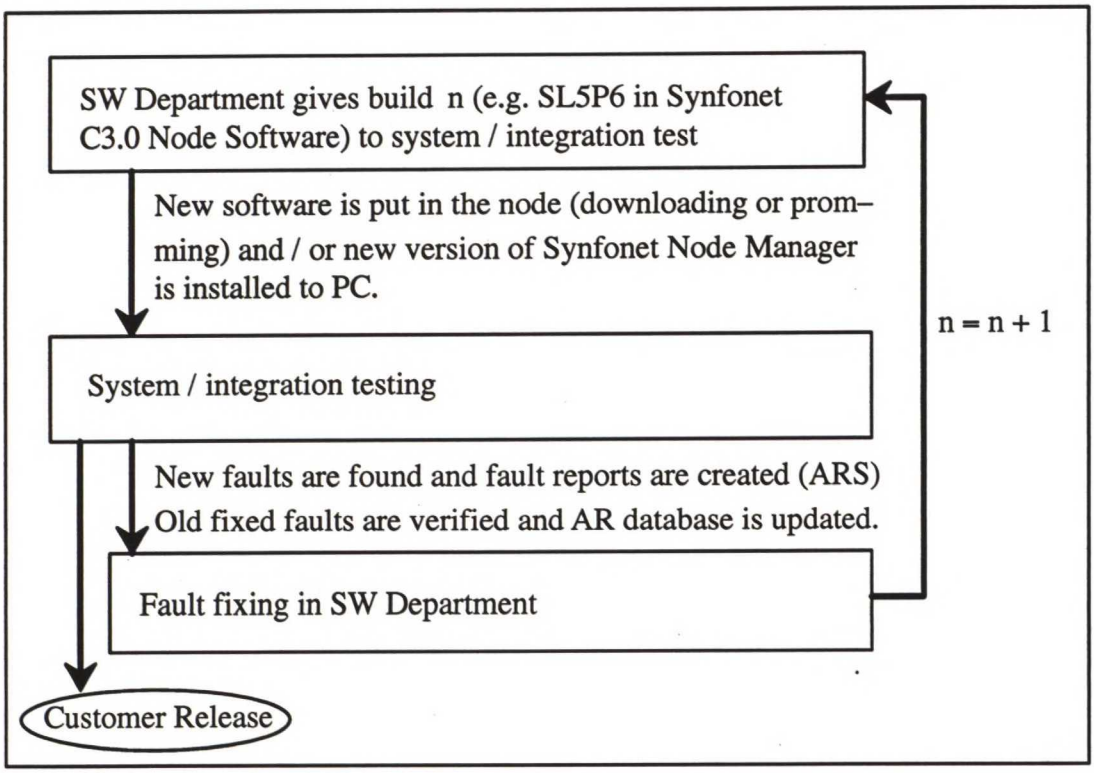


Figure 16. *Synfonet (node) software testing cycle.*

6 INTRODUCTION TO SOFTWARE RELIABILITY

6.1 Overview

Many attributes can be used to characterize the quality of a software product, such as performance, reliability, maintainability, comprehensibility, testability and reusability [8]. Measuring the reliability of a program has proved to be a challenging task. Even though, the importance of software in all aspects of modern life demands continuing research to discover effective methods of developing reliable software and assessing their reliability. Software reliability can be defined [9] as

- The probability that software will not cause the failure of a system for specified time under specified conditions
- The probability is the function of the inputs to, and use of, the system as well as a function of the existence of faults in the software
- The inputs of the system determine whether existing faults, if any are encountered

Software error, fault and failure are among the most important software reliability terms. These terms are defined in Figure 17.

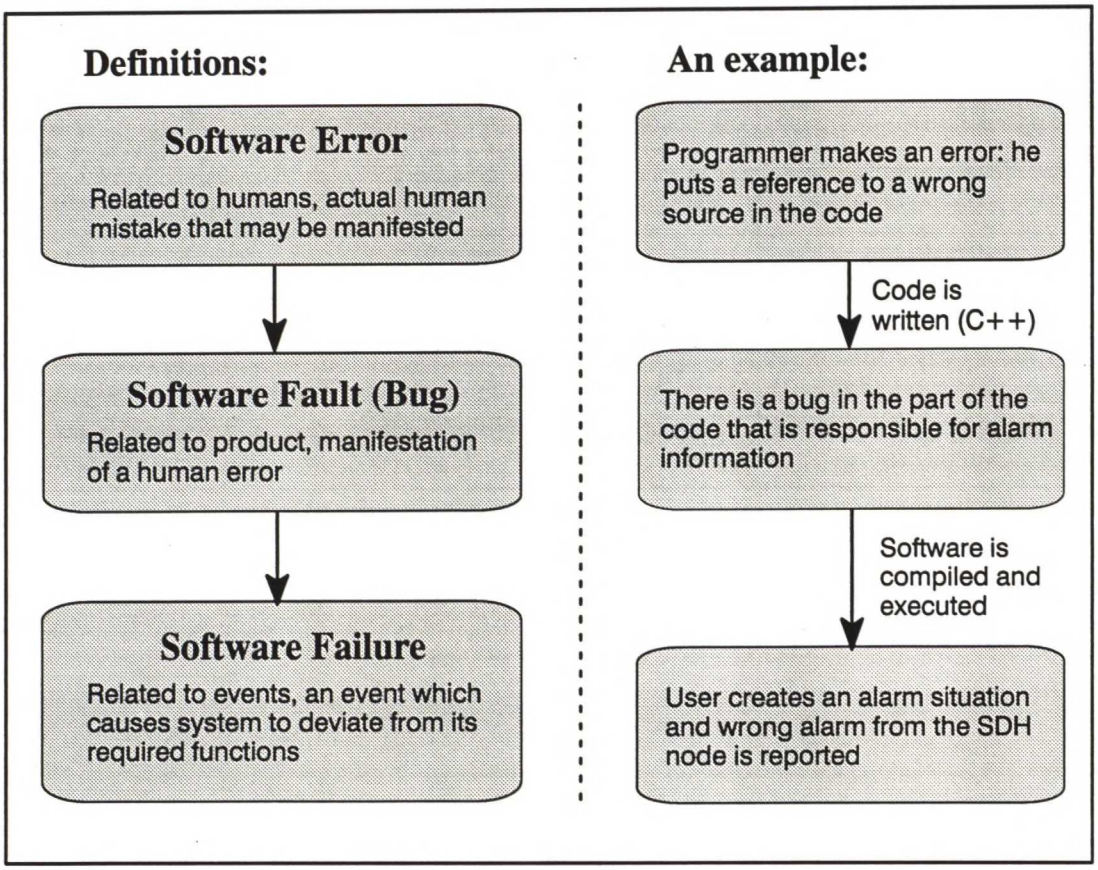


Figure 17. *Software error, fault and failure.*

There are also many other terms that are commonly used. Table 4 represents some important concepts and definitions in software reliability analysis.

Table 4. *Some software reliability terms defined [9].*

SW Reliability Term	***Definition***
Failure Modes	Events related to failures normally described in terms of the system, e.g. Loss of communication in the transmission network.
Hazard	A situation that can lead to an accident, danger to life, safety or monetary loss. A fatal fault in node software is a hazard. The failure mode is the crash of the system and thus loss of data traffic.
Defect	A fault which is found prior to the software becoming operational.
Corrective Action	A maintenance action that repairs a software fault. Corrective action can be done incorrectly and it may introduce new faults.
Preventive Maintenance	A maintenance action that makes a given segment of code more fault tolerant or robust.
Maintenance	A life cycle phase which includes corrective action, preventive maintenance and enhancements.
Maintainability	For hardware maintainability is downtime and restore time, for software it is corrective action time.
MTTF	Mean Time To Failure – The mean time to the next software failure.
Operational / Usage Time	During development this is the total test time. During operation, this is the total usage time of the software.
Execution Time	The CPU time executed during software testing and / or usage.
Calendar Time	The total time that has expired while the software has been operational (typically 24 h per day).
Mean Repair Time	The mean time to isolate, repair, and checkout a corrective action.
Mean Turnaround Time	The mean time to administer, isolate, repair, checkout, re–configure and distribute one or more corrective actions.
Software Environment	Associated hardware, software, end users and probability of being used.

Software faults are caused by errors made during [9]:

- requirements definition and analysis,
- design,
- implementation (code writing), or
- maintenance.

Requirements errors occur when some requirements are not implemented, implemented incorrectly, or not developed for full set of conditions. Design errors typically occur when design is not complete, not implemented for full range of inputs, or design implementation is not according to specifications. Typical implementation (coding) errors are initialization errors, storing errors, faulty parameter passing, misuse of variables and missing code. Maintenance errors are e.g. introducing new faults in corrective action, preventive action, or enhancements and partial instead of complete corrective action.

One good example of the behaviour of faulty software is Ariane–5 disaster. Four scientific satellites worth $500 million were destroyed. The loss was due to specification and design errors in the software of the rocket guidance system. The quality of the testing was not adequate to detect the potential failure. This is a good example, why software reliability is a very important concept.

The concept of software reliability is different from hardware reliability. Software components do not wear out, and unchanged software will always behave in the same way under similar external conditions. But in the reality, the division between hardware and software reliability is somewhat artificial [10]. It is possible to combine hardware and software component reliabilities to get system reliability. The source of failures in software is design faults and in the hardware it usually is physical deterioration. If a software (design) defect is properly fixed, it is in general fixed for all time. Software failure usually occurs only when a program is executed in an environment that it is not developed or tested for. The "design reliability" is thus a very important concept in software development. Software reliability tends to change continually during test periods. This happens when new problems are introduced by writing new code or when repair action removes (adds) problems in the code. It can be said that hardware reliability has a much greater tendency to stabilize towards a constant value.

Reliability quantities are usually defined with respect to time [10]. The execution time τ for a program is the time that is actually spent by processors in executing the instructions of the program. Calendar time t is the time we normally experience, e.g. measured in weeks since the beginning of the integration test period. Clock time represents the elapsed time from start to end of program execution on a running computer / system. As an example of these three types of time, consider Synfonet Node Manager that is tested by a test engineer in an SDH network. In one week of calendar time, there may be 35 h of clock time during which the system is running. There might be 20 h of execution time for SNM program itself. When we talk about execution or clock time, any down time of the system (in our case e.g. the time when the PC is switched off) is excluded.

There are four general ways of characterizing failure occurrences in time [10]:

- time of failure,
- time interval between failures,
- cumulative failures experienced up to a given time, and
- failures experienced in a time interval.

Two last ones are probably the best ways to measure the failures during the SDH software integration / system testing.

Failure behaviour is affected by two principal factors:

- the number of faults in the software being executed, and
- the execution environment or operational profile of execution.

The number of faults in the software is the difference between the number introduced and the number removed.

6.2 Software Reliability Models

A good software reliability model has many important characteristics. It [10, p.19–20]:

- gives good predictions of future failure behaviour,
- computes useful quantities,
- is simple,
- is widely applicable, and
- is based on sound assumptions.

Software reliability models are based on a stable program executing in a constant environment. This means that neither the code nor the operational profile are changing. This is usually not the case in SDH product development. If the profile and environment change, the modeling is handled in a piecewise fashion.

A good model enhances communication on a project and provides a common framework of understanding for the software development process [10]. It also enhances visibility to management and other interested parties, e.g. marketing department.

Musa, Iannino and Okumoto [10] suggest two models for software reliability analysis. Those models are

- the basic execution time model, and
- the logarithmic Poisson execution time model.

6.2.1 Execution Time Component

The execution time component for both models assumes that failures occur as a random process, in fact, a nonhomogeneous Poisson process [v]. The term "nonhomogeneous" means that the characteristics of the probability distribution that make up the random process vary with time. This is natural behaviour, because faults are both being introduced and removed as time passes. In the Poisson distribution, the probability that n failures occur is

$$P(X=n) = (\lambda^n / n!) \exp(-\lambda). \quad (1)$$

λ is the parameter of the Poisson distribution, it is both the mean and the variance of the distribution.

The difference between the basic execution time model and the logarithmic Poisson execution time model is best described in terms of slope or decrement in failure intensity per failure experienced. The difference is best described in Figure 18. In the basic model, the decrement in the failure intensity function remains constant. In the logarithmic Poisson model the failure intensity drop is logarithmic.

The failure intensity λ as a function of failures found for the basic model [10] is

$$\lambda(\mu) = \lambda_0 (1 - \mu/\nu_0). \quad (2)$$

λ_0 is the initial failure intensity (e.g. 20 failures per week), μ is the number of failures experienced, and ν_0 is the total number of failures in the software product.

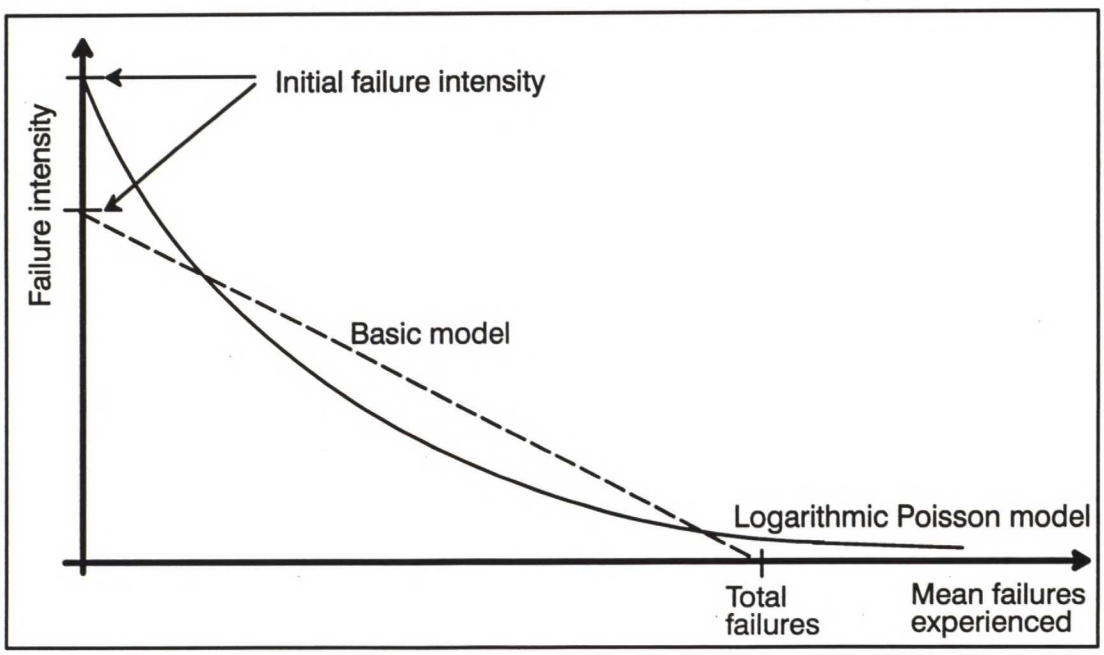


Figure 18. *Failure intensity functions.*

The slope of failure intensity, $d\lambda/d\mu$, for the basic model is [10]

$$d\lambda/d\mu = -\lambda_0/\nu_0. \quad (3)$$

The failure intensity for the logarithmic Poisson model is [10, p. 34]

$$\lambda(\mu) = \lambda_0 \exp(-\theta\mu). \quad (4)$$

θ is called the failure intensity decay parameter. If we plot the natural logarithm of the failure intensity (see (4)) against the mean failures experienced, θ is the magnitude of the slope of the line plotted. It represents the relative change of failure intensity per failure experienced.

The derivate of the failure intensity for the logarithmic Poisson model is

$$d\lambda/d\mu = -\theta\lambda_0 \exp(-\theta\mu) = -\theta\lambda. \quad (5)$$

The basic model implies a uniform operational profile. The failure intensity curve for highly nonuniform operational profile is convex [10, p.35]. The logarithmic Poisson model probably is superior for highly nonuniform operational profiles, where some functions are executed more often than others. In the end (see Figure 18) the failure intensity in the logarithmic Poisson model decreases very slowly. This is because of the very infrequent execution of the input states that still contain faults. This may sound unrealistic when talking about system test.

Almost each failure found will generate some repair activity and the result of the repair activity is a decrement in the failure intensity. Another question is, does the repair remove the fault perfectly or does the repair generate new faults. The two models do not make any assumption about the quality of the repair process and they allow the possible introduction of new faults during repair. Both models also assume that the correction of the faults is immediate. This is not the case in the real life. Usually the delay is between two days and two weeks. It is also possible that a potential failure is masked by a previous failure. The failures that mask other failures should be removed from the program before starting the system test and the usage of reliability models.

Let the execution time be denoted by τ. The number of failures experienced can be written as a function of the execution time [10, p. 37]:

$$\mu(\tau) = \nu_0 (1 - \exp(-\lambda_0 \tau / \nu_0)). \quad (6)$$

And for the logarithmic Poisson model, we have

$$\mu(\tau) = 1/\theta \ln(\lambda_0 \theta \tau + 1). \quad (7)$$

The execution time component for both models is characterized by two parameters: initial failure intensity and failure intensity change. Initial failure intensity is described using the same parameter, λ_0, for both models. Failure intensity change for the basic model is described using the number of total failures, ν_0. In the logarithmic Poisson model, the failure intensity decay parameter, θ, is used.

Model parameters can be divided into two categories: execution time component and calendar time component parameters. If information about calendar time (when the software product is ready), resource expenditures or costs are needed, the calendar time component is important. The values of two parameters must be determined for the execution time component. The initial failure intensity λ_0 is needed for both models. Additionally, total failures experienced ν_0 is needed for the basic model, and failure intensity decay parameter θ is needed for the logarithmic Poisson model. It is possible to predict (determine before program execution) the parameter values for the basic model using information about the program structure. After the program has executed and failure data is available, it is possible to estimate those parameters.

If a software product has been released and is operational, no new features are added or repairs made between releases, the failure intensity will be constant. Then both models reduce to homogeneous Poisson process with the failure intensity as a parameter [10, p. 50]. The failure intervals follow an exponential distribution and the reliability R can be defined this way:

$$R(\tau) = \exp(-\lambda\tau). \tag{8}$$

λ is the failure intensity (constant) and τ is the execution time. The reliability R is the probability of no failures in a period of execution time τ.

6.2.2 Calendar Time Component

The calendar time component relates execution time and calendar time by determining the calendar time to execution time ratio at any given point in time [10, p.50]. The ratio is based on the constraints that are involved in applying resources to a project. The calendar time component is very important during software testing and repair phases. When the failure intensity is constant and no repairs are made, the ratio between execution time and calendar time is constant.

When integration / system test is done, the rate of testing is constrained by the failure identification (test team) personnel, failure correction (debugging) personnel and the computer time available. The last constraint can be understood as the equipment available (nodes, analyzers or PCs) during integration / system test.

At the start of testing the test team identifies a large number of failures (e.g. 10 in a day). Testing must sometimes be limited to let the people who are fixing the faults keep up with the load. When testing goes on, the intervals between failures become longer. It is possible that the failure correction personnel is not completely employed with failure correction work. Then the test team becomes the bottle-neck.

***Table 5.** Calendar time component parameters and resources [10, p. 52].*

	Usage parameters – Requirements per		Planned parameters	
Resource	CPU hr τ	Failure μ	Quantities available	Utilizations
Failure identification personnel	θ_I	μ_I	P_I	1
Failure correction personnel	0	μ_F	P_F	ρ_F
Computer time	θ_C	μ_C	P_C	ρ_C

Let χ_r, $r \in \{I, F, C\}$, be the usage of resource r. Then [10]

$$\chi_r = \theta_r \tau + \mu_r \mu. \tag{9}$$

θ_r is the resource usage per CPU hour. It is nonzero for failure identification personnel (θ_I) and computer time (θ_C). μ_r is the resource usage per failure, τ is the number of CPU hours, and μ is the mean failures experienced. The resource usage μ_r is nonzero for failure identification personnel (μ_I), failure correction personnel (μ_F), and computer time (μ_C).

Let's think about the SDH system test team testing SNM C2.1 for three (CPU) hours. The work effort related to failures experienced is dependent on the number of failures. A failure report (an Action Request) has to be written for each failure found. Each failure has to be checked carefully (is it a real failure, is the failure already reported, is it reproducible?). Consider the test team founds 8 failures, the effort required per hour of execution time is 5 person hours, and each failure requires 2 hours on the average to verify and determine its nature. Then the total failure identification effort required (χ_r), using equation (10), is 5 x 3 + 2 x 8 = 31 person hours.

For failure correction team, the resources required are dependent only on the mean failures experienced. In the testing phase, computer time required per unit execution time is normally greater than 1. Additional programs are needed during the test, such as test drivers, recording routines, and separate test software tools.

The change in the resource usage per unit of execution time can be obtained by differentiating Equation (9):

$$d\chi_r / d\tau = \theta_r + \mu_r \, d\mu/d\tau = \theta_r + \mu_r \lambda. \tag{10}$$

Because the failure intensity decreases with testing, the effort used per hour of execution time tends to decrease with testing. The change in the resource usage per failure can be calculated differentiating Equation (9) with respect to failures:

$$d\chi_r / d\mu = \mu_r + \theta_r \, d\tau/d\mu. \tag{11}$$

The instantaneous calendar time to execution time relationship can be obtained by dividing the resource usage rate of the limiting resource by the constant quantity of resources available that can be utilized [10, p. 55]. Let t be calendar time, P_r represent resources available, and ϱ_r be the utilization. Then

$$dt/d\tau = (1/P_r\varrho_r)\, d\chi_r/d\tau = (\theta_r + \mu_r\lambda)/P_r\varrho_r. \tag{12}$$

Two categories of parameters must be established for the calendar time component: planned parameters and resource usage [10, p. 57]. Planned parameters are established by managerial decisions on the project or by project conditions and policies. Resource quantity parameters include e.g. the size of the system test team and the number of failure correction personnel employed on the project.

6.3 Estimation

Maximum likelihood estimation can be used when the form of the failure intensity is known. The maximum likelihood estimate $\chi_{ML} = \chi_{ML}(k_1, k_2, ..., k_N)$ of X in terms of the observed subinterval failure intensities $k_1, k_2, ..., k_N$ is by definition the value of X that maximizes the probability of having observed $k_1, k_2, ..., k_N$ [11, p.75]. The maximization is subject to the constraint that $\chi_{ML} \in \mathfrak{X}$; i.e. the estimate must lie in the space of possible parameter values. $\lambda_\sigma(\mathbf{X})$ is the intensity function.

$$P_r[N_{t_{i-1},t_i} = k_i;\ i=1,...,N \mid \mathbf{X}] = \prod_{i=1}^{N} \left\{ [(k_i)!]^{-1} \left(\int_{t_{i-1}}^{t_i} \lambda_\sigma(\mathbf{X})d\sigma\right)^{k_i} \exp\left[-\int_{t_{i-1}}^{t_i} \lambda_\sigma(\mathbf{X})d\sigma\right] \right\} \tag{13}$$

The value of **X** that maximizes this probability also maximizes its logarithm, because the logarithm is a monotonically increasing function of its argument. We can get the logarithmic likelihood function $l(\mathbf{X})$ by taking the natural logarithm of (13):

$$l(\mathbf{X}) = -\int_{t_0}^{T} \lambda_\sigma(\mathbf{X})d\sigma + \sum_{i=1}^{N} k_i \ln\left(\int_{t_{i-1}}^{t_i} \lambda_\sigma(\mathbf{X})d\sigma\right). \tag{14}$$

The maximum likelihood estimate χ_{ML} of X can be calculated by maximizing l(X), the maximization is subject to the constraint $X \in \mathfrak{X}$. The maximization in done by differentiating l(X).

Estimation is usually accomplished in system test or operational phase. It is usually more accurate than prediction. The accuracy of the estimation increases with the size of the sample of failures. It is possible to estimate confidence intervals to characterize the accuracy. The confidence intervals represent the range of parameters that could possibly explain the data experienced at some level of confidence (e.g. 95 %).

6.4 Collecting Data to the Models

Careful planning and organization are necessary if we want to collect data of high quality. The data collection should happen in real time, e.g. every day, twice a week or once a week. Valuable data is lost if this is not done. It is not possible to get sufficient data afterwards.

Action Request database is the most important source of information and the data should be utilized as efficiently as possible. It is important that all necessary information is put in the failure reports. The expert opinions given by the members of the system test team must not be forgotten. Those opinions often give the most realistic picture of the state of the software to the management.

Musa [10] believes that the following approach to the data collection is fruitful:

- Obtain the motivated participation of the data takers.
- Make the collection mechanism as easy as possible.
- Collect and scrub the data in real time.
- Provide feedback of results obtained from the data on a regular and timely basis.

A suggestion for weekly failure data collection is presented in Section 7.8. It includes the follow–up of Action Request database and system test group expert opinions.

7 RELIABILITY ANALYSIS OF SYNFONET SDH SOFTWARE

7.1 Introduction

Synfonet SDH equipment is a complex software and hardware system. The software in Synfonet has a very important role. The software can basically be divided in two parts: the node and the node manager software (Synfonet Node Manager). The most important role of the node software is to manage the hardware and take care of transmission. Hardware in this case means e.g. ASICs, laser transmitters / receivers and node backplanes. The most important task of the Windows based Synfonet Node Manager is to install, configure and manage the SDH nodes.

7.2 Reliability of the SDH Equipment

Different people have different opinions on the reliability of a software product. For example, marketing people, test department people, code writers and customers have their own point of view on the product reliability. One definition for good quality SDH equipment could be that the transmission and network management function the way the customer assumes them to function. One good meter of software reliability is that the software behaviour is predictable, i.e. the manufacturer knows how well the product will function in the future. Of course, the software development model, design, code writing, the structure of the code and effective testing are very important factors affecting the software reliability.

An operational profile can be defined as the probability density function over the whole input state that best represents how the inputs would be selected during the software life–time. An operational profile is thus a kind of guess what will occur in the future. It gives the usage frequencies of different functions in the software product. Suppose those parts of the software, that the customer uses, work perfectly. Then the customer sees the reliability of the software product as 100 per cent.

7.3 Testing in Practice

At the moment, the SDH system test group consists of an engineering manager and several system test engineers. This group is responsible for writing system test specifications for Synfonet node software and Synfonet Node Manager, making integration testing, and making the final system tests. Ideal situation is that when the system tests begin, the software should not contain any known fatal / major faults. But in the real life, this is not always possible.

The creative and experience based random testing is probably the most important part of the software testing. This testing is done before formal specifications based system testing. This testing gives an answer to the question, whether the equipment works correctly and whether it is usable. Based on experience, it can be said, that the finding of failures is very much based on the test effort done outside the specifications and the skill of the test engineers.

7.4 Failure Descriptions in Synfonet Products

The definition of failures sharpens the system definition by providing the perspective of what the system should not be doing [10, p. 78]. In writing requirements, a positive specification of a system is created. In defining failures, a negative specification is created. It is very useful to identify as many possible failure types as possible in SDH equipment. The failure severity classes are defined in Chapter 5.3.2.

FMEA

Failure Mode and Effects Analysis (FMEA) [12] activities are designed to: 1) recognize and evaluate the potential failure of a (software) product and its effects; 2) identify actions which could eliminate or reduce the chance of the potential failure occurring; and 3) document the process. The intent of FMEA activities is to enhance the design process and provide greater assurance and satisfaction to the customer. The failure type analysis made for node software and node manager software in this thesis is a partial FMEA for Synfonet software.

7.4.1 Node Software Failure Types

Failures in the node software can be caused by different parts of the code. The most important possible failure types for the node software are:

- transmission does not work correctly (bit errors, transmission not working at all)
- crash of a card (plug–in unit) / many cards
  - recoverable (transmission ok after e.g. 30 seconds) / non–recoverable (e.g. reset loop)
- wrong alarm (e.g. Alarm Indication Signal without acceptable reason) or no alarm when needed
- event sending not working correctly / node sends wrong kind of events
- protection switches not working correctly
  - transmission path protection (SNC = Sub–Network Connection)
  - hardware protection (UP = Unit Protection: CU, SSW and TA protection)
- functionality / functionalities not according to specifications (e.g. missing functionalities)
- node installation / configuration problems (e.g. crashing cards)
- node update (release changing) problems
  - software update e.g. from C2.1 N to C2.1 N+1
  - system release upgrade from C2.0 to C2.1
- backup problems
- problems related to Q3 Stack (communicating with the node)

- message routing problems
- randomly changing / disappearing settings in the node
- wrong node state information (e.g. performance monitoring)
- memory leaks
- operating speed too slow
- incompatibility with other equipment
    - other manufacturers SDH equipment
    - PDH equipment
    - previous release SDH equipment

7.4.2 Synfonet Node Manager Failure Types

The most important requirements for Synfonet Node Manager are:

- SNM must not crash in any situation
- SNM must not give any wrong information to the user (alarms, etc.)
- SNM must not do anything extra unspecified by the user

Failure types in SNM can be listed this way:

- crashes (reproducible / random)
    - General Protection Faults (GPFs) in Microsoft Windows 3.x
    - Illegal Operations in Microsoft Windows 95
    - Application Errors in Microsoft Windows NT
- wrong / missing alarm information
- wrong information about the state of the node (e.g. synchronization window information is incorrect)
- making wrong settings to the node or SNM does not make some settings
- not possible to make settings that are in the specifications
- SNM installation to a PC
- version changing problems (SNM C2.0 –> C2.1, C2.1 N –> C2.1 N+1)
- node database problems
- incompatibility with other PC programs (e.g. NMS/10)
- wrong kind of windows, wrong texts, wrong colours
- connection problems to the node
- operating speed too slow
- memory leaks

7.5 Comparison between Releases

Nokia released its first SDH equipment in 1993. The equipment was called SLA4 (Synchrone Leitungsausrüstung, Synchronous Line Equipment). The most important customer for SLA4 was Deutsche Bundespost in Germany. The release supported two node types: SLX1/4 multiplexer and SLR4 regenerator. The interfaces in SLA4 were optical STM–4 and electrical STM–1E / 140M. SLA–4 was designed for point–to–point connections.

The first flexible SDH release was Synfonet C1.0 in 1994. The node types in Synfonet C1.0 were terminal multiplexer, add/drop multiplexer and digital cross connect node. The node management was handled using Q1 management interface. In addition to SLA4, Synfonet C1.0 also supported STM–1 and 2M interfaces. Cross connections could be created at VC–12 level.

The main change between Synfonet C1.0 and Synfonet C2.0 was Q3 (ethernet) management interface. Also, cross connections could be protected (SNC protection).

Synfonet release C2.1 / C3.0 brought many major changes to C2.0. The release C3.0 uses new plug–in units (HW2, Hardware 2). This makes the software downloading possible; in C3.0 the changing of memory circuits is not needed any more. Otherwise the functionality of Synfonet C3.0 is identical to C2.1. Making auxiliary (DCC) connections was one new option. Cross connections could be created also at VC–4 level, and 140M interface (not in C2.0) was taken in use. The most challenging task for C2.1 / C3.0 was hardware unit protection for CU, SSW and TA. This also created many new failure situations during the integration / system test.

The future releases C2.20 / C3.20 will include new functionality, such as 34M interface, VC–2 and VC–3 level cross connections and regenerator as a node type.

The node management is based on Synfonet Node Manager (SNM) using Windows 3.x platform. SNM C2.1 is also available as Windows NT version. The future platform will be Windows 95 / NT beginning with SNM C2.20. The first NMS/10 (TMS–NetMan) was released for Synfonet C2.0 network management. NMS/100 was for the first time released for Synfonet C2.1 / C3.0 node equipment.

7.6 Failures in the Node Software

7.6.1 Failure Distributions

It is possible to classify the failure data in different releases according to many criteria. In Synfonet node software the found failures are classified by severity in Figure 19. The number of failures varies a lot in different releases. There was twice as much failures in Synfonet C1.0 as in Synfonet C2.0. The portion of fatal and major faults is between 62 and 76 per cent. The portion of major faults seems to be the most stable; it is between 42 and 46 per cent in different releases. The number of cosmetic and enhancement ARs shows a growing trend. But their portion is less than eight per cent in each release.

Synfonet C2.0 seems to be different from C1.0 and C2.1/C3.0. One reason for this is that the total number of failures is very much smaller for Synfonet C2.0.

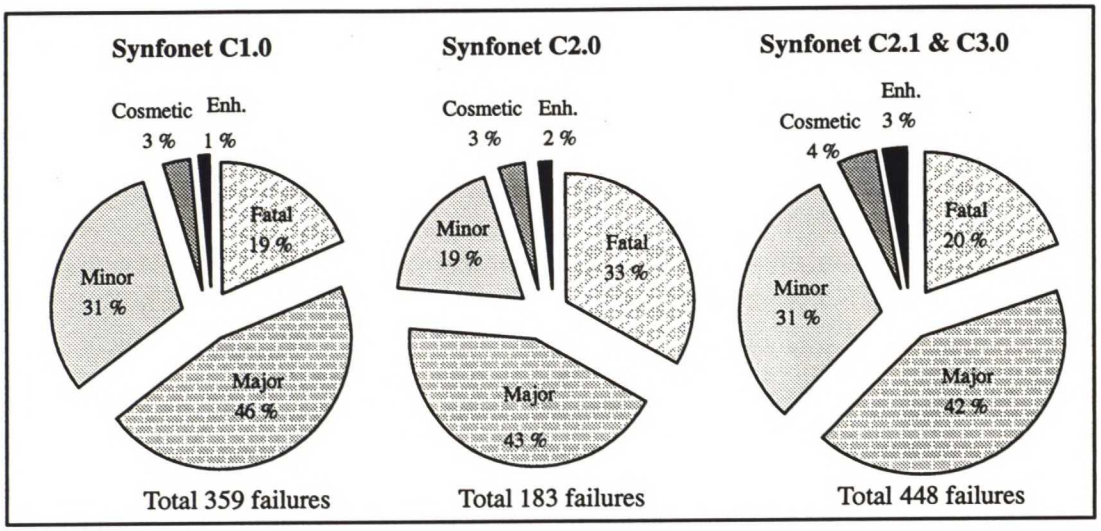


Figure 19. *Synfonet node software failure severity distributions.*

Over 90 per cent of the failures in Synfonet node software are found in Application Software (ASW), Node Functional Layer (NFL), and Transmission Hardware Drivers (THD). For closer details of these components, see 3.6.1. This is why the failure distributions are presented for these three components in Figure 20. Over 50 per cent of the fault reports are created for ASW in each release. ASW is the node software layer that communicates with the node management applications. THD has the second biggest number of fault reports in Synfonet C1.0 and C2.1/C3.0 node software. Synfonet C2.0 seems to be the one that differs from C1.0 and C2.1/C3.0.

A more specific statistical analysis of the failure distributions is done in Section 7.8. Contingency tables are created and χ^2 test is used.

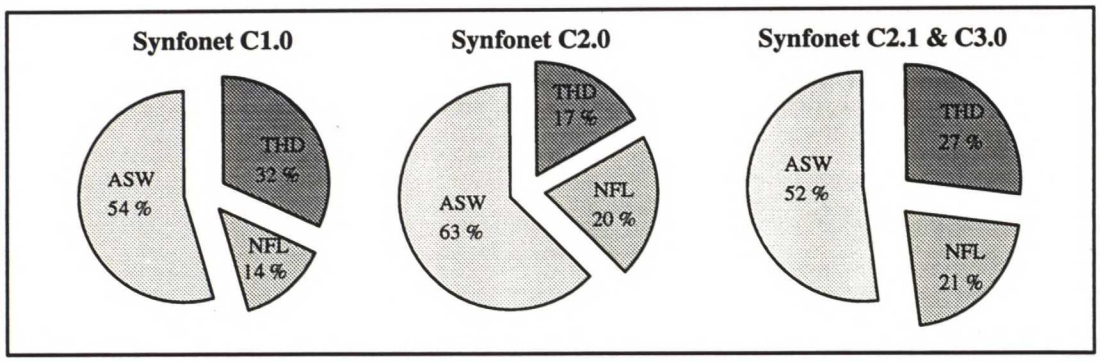


Figure 20. *Synfonet node software failure distributions.*

7.6.2 Cumulative Failures and Failure Frequencies

Failure statistics have been taken manually from the Action Request database. The cumulative number of failures for Synfonet C1.0, C2.0 and C2.1 / C3.0 node software is displayed in Figures 22–24. The failure curves plotted in the same picture is represented in Figure 21. Node software testing is done in two groups: software integration group and system test group. System test group also makes integration testing before the real system test begins. Synfonet C1.0 and C2.0 node software cumulative failure plots include the fault reports made by the system test group only. Synfonet C2.1 / C3.0 node software cumulative failure plot also includes the fault reports made by the software integration group. These plots include failures in all severity classes (from enhancement to fatal) and the basic time interval in displaying data is one week in calendar time.

The plots begin from the time the first ARs of the node software are created. In the beginning, the node software is so unstable that only few failures are found. Most of the time is spent setting the system up and trying to keep it working. Approximately from five to ten fatal faults prevent finding the other faults. Examples of that kind of fatal faults are inability to install some node type, continuously crashing card types and inability to get the signal through the equipment. About 12 to 18 weeks after the beginning of the integration testing, the failure finding rate is accelerating. It will be highest about 20 weeks after the beginning of the testing. It can be seen in Figure 21 that the growth of the number of failures stabilizes after 30 weeks in Synfonet C2.0 and C2.1/C3.0. Synfonet C1.0 failure amount stabilizes after 40 weeks. This is typical for the first release.

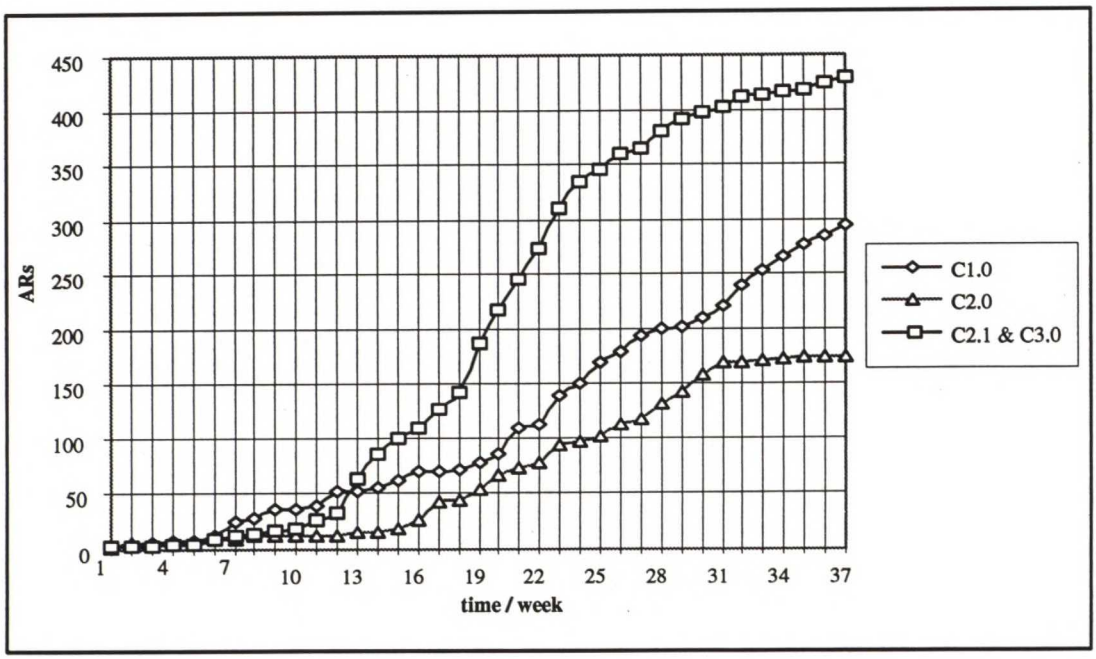


Figure 21. *Cumulative number of node software failures (ARs) in different Synfonet releases.*

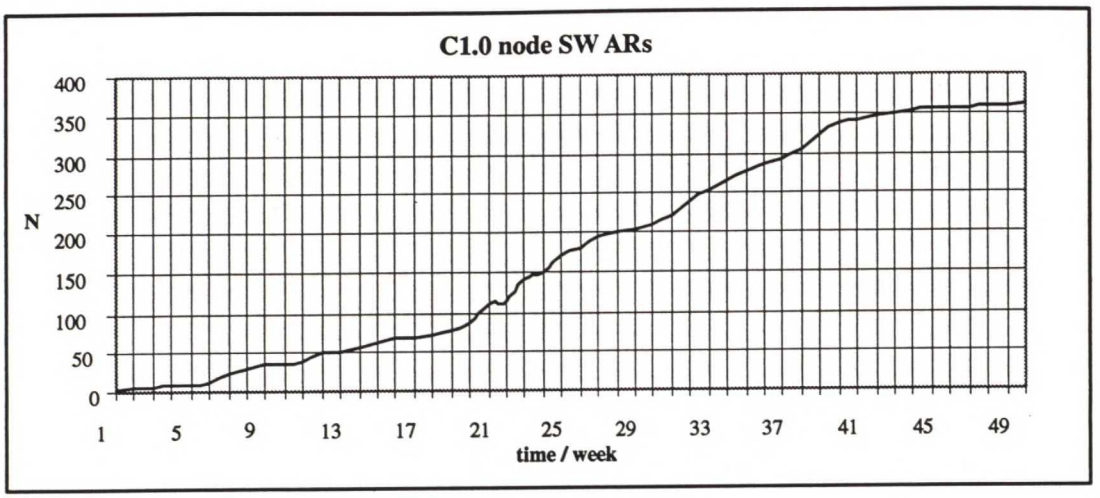


Figure 22. *Cumulative number of failures (ARs) in Synfonet C1.0 node software.*

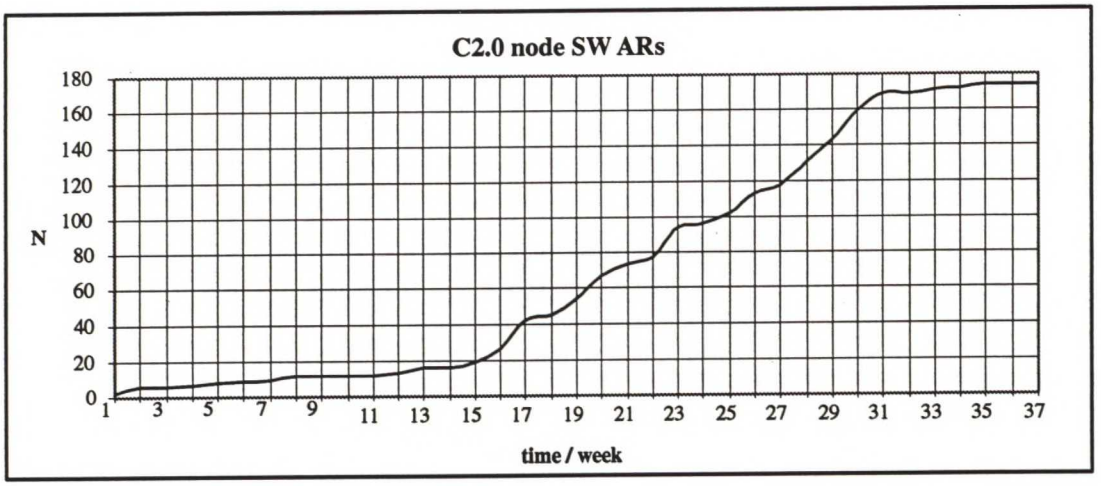


Figure 23. *Cumulative number of failures (ARs) in Synfonet C2.0 node software.*

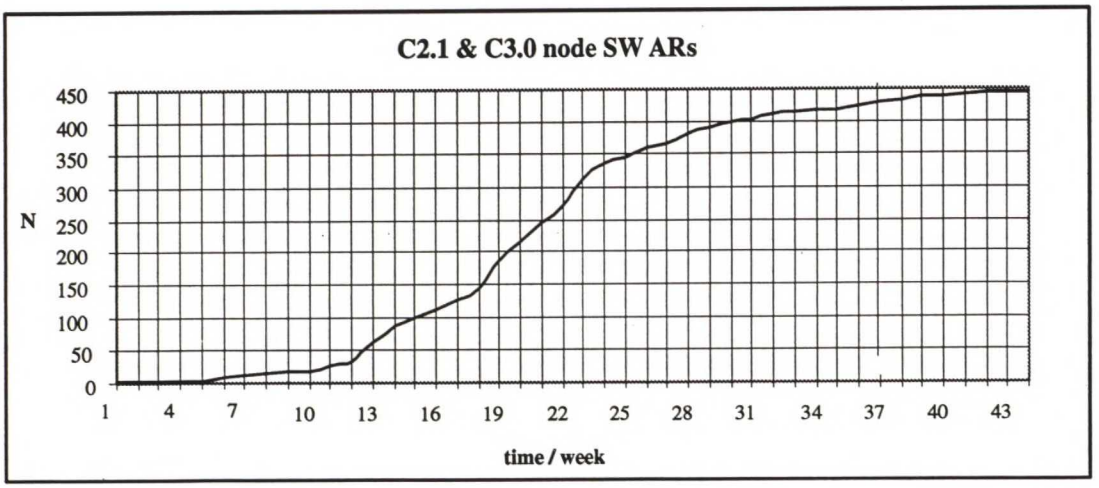


Figure 24. *Cumulative number of failures (ARs) in Synfonet C2.1 & C3.0 node software.*

It is interesting to plot the cumulative number of fatal and major ARs. This is done for Synfonet C2.1 / C3.0 in Figure 25. If Figures 24 and 25 are compared, the same kind of format is recognized. One reason for this is that the fatal and major ARs represent 62 per cent of the total failures.

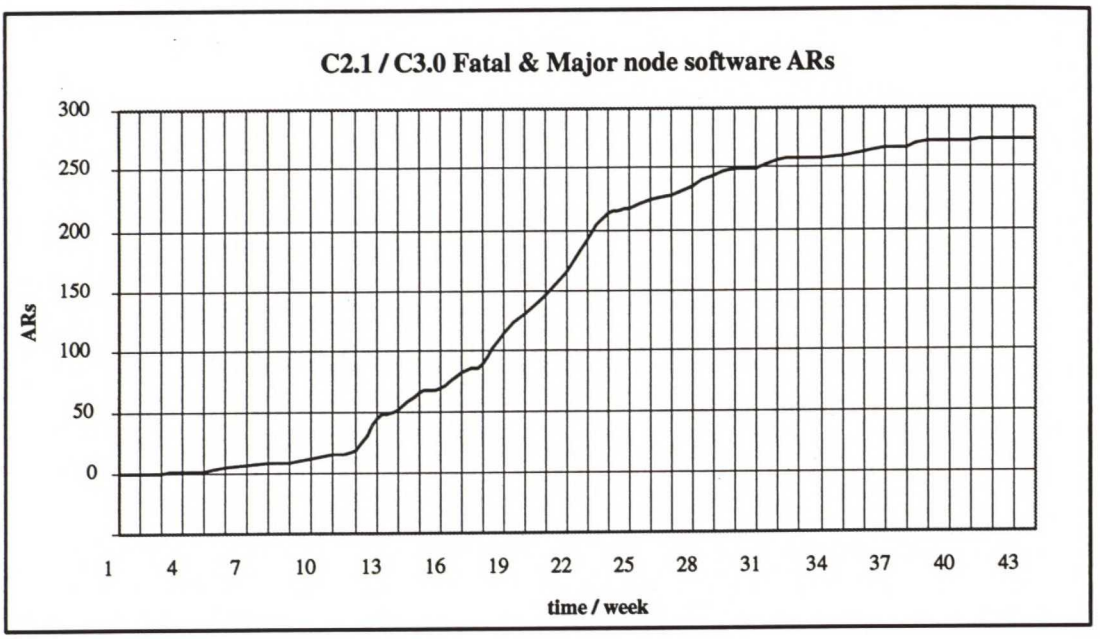


Figure 25. *Cumulative number of major and fatal failures (ARs) in Synfonet C2.1 & C3.0 node software.*

The failure intensity is the change in the number of cumulative failures during a week. This is plotted for Synfonet C1.0, C2.0, and C2.1 / C3.0 node software in Figures 26 – 28. All failure intensity curves are smoothed. Every plotted data point is an average of six data points.

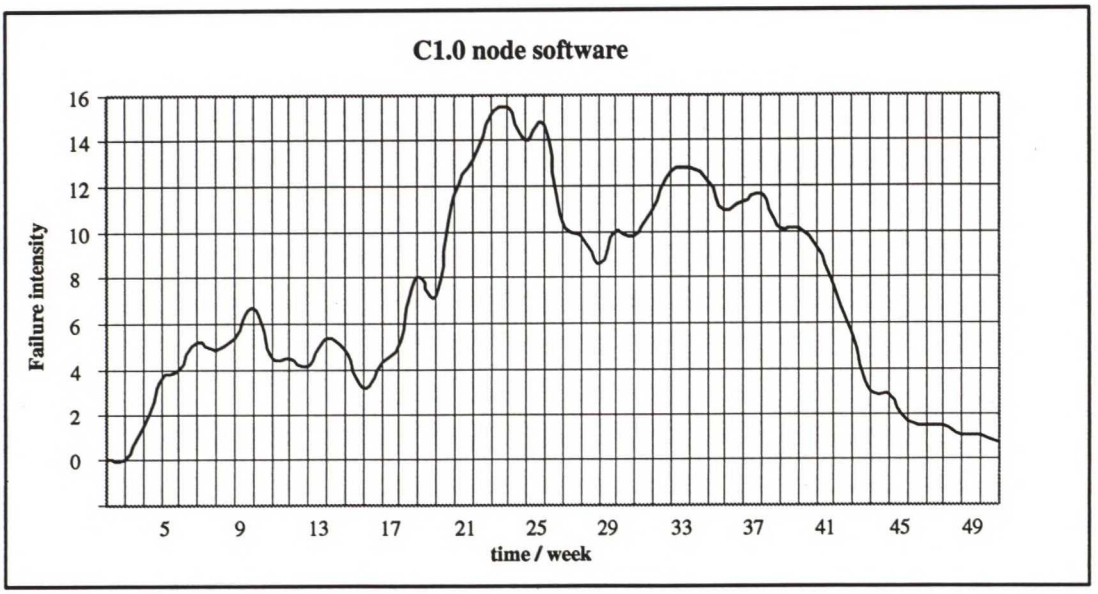


Figure 26. *Synfonet C1.0 node software weekly failure intensity.*

The plots for Synfonet C1.0 and C2.0 are quite clearly step functions. This is because of the nature of the test. The node software to be tested had to be changed by burning the software to the memory circuits. The update of the software took a long time. That's why the test group did not get new releases very often. Some parts of the plots imply the adding of new functionalities and thus an increase in the failure intensity. Such a point can be found in Figure 27 after 25 weeks.

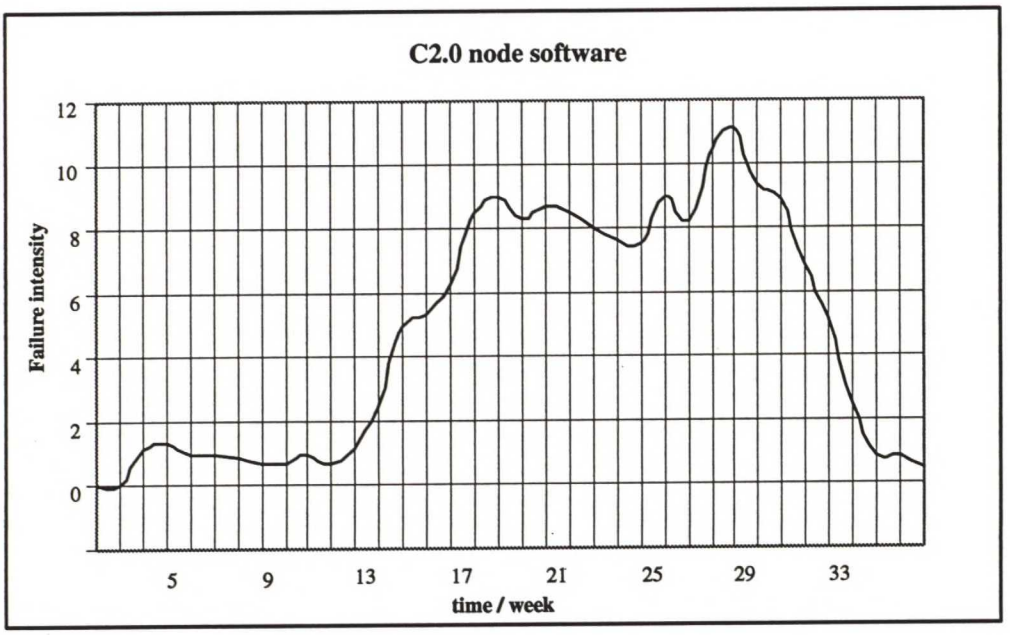


Figure 27. *Synfonet C2.0 node software weekly failure intensity.*

Synfonet C2.1 / C3.0 node software behaviour seems to be more stable. There is only one maximum point in Figure 28 after 22 weeks. The curve is also very symmetric. It can be said that the behaviour of C2.1 / C3.0 node software has been more predictable.

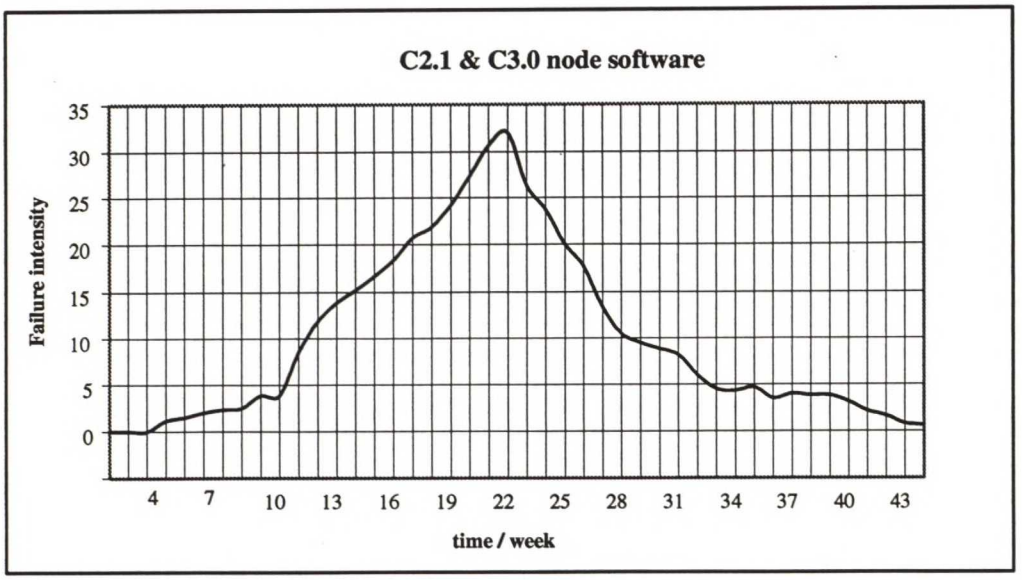


Figure 28. *Synfonet C2.1 & C3.0 node software weekly failure intensity.*

It is very interesting to note that Figures 26–28 representing failure intensities do not have the format of Figure 18. The failure intensity does not have its maximum value in the beginning of the test. The maximum value is reached after 20 weeks. Why does the node software failure intensity not follow the theory? This is because the failure data contains both integration test and system test data. In the beginning of the test the software is so unstable that it can't be efficiently tested. It is not possible to find dozens of functional failures in a piece of software that crashes every ten minutes. That's why we must be critical when interpreting the results. We must know what kind of process produces the data.

Failure intensities plotted against the cumulative number of ARs are displayed in Figures 29–31. The axis have been selected so that only the decreasing trend of failure intensity is displayed.

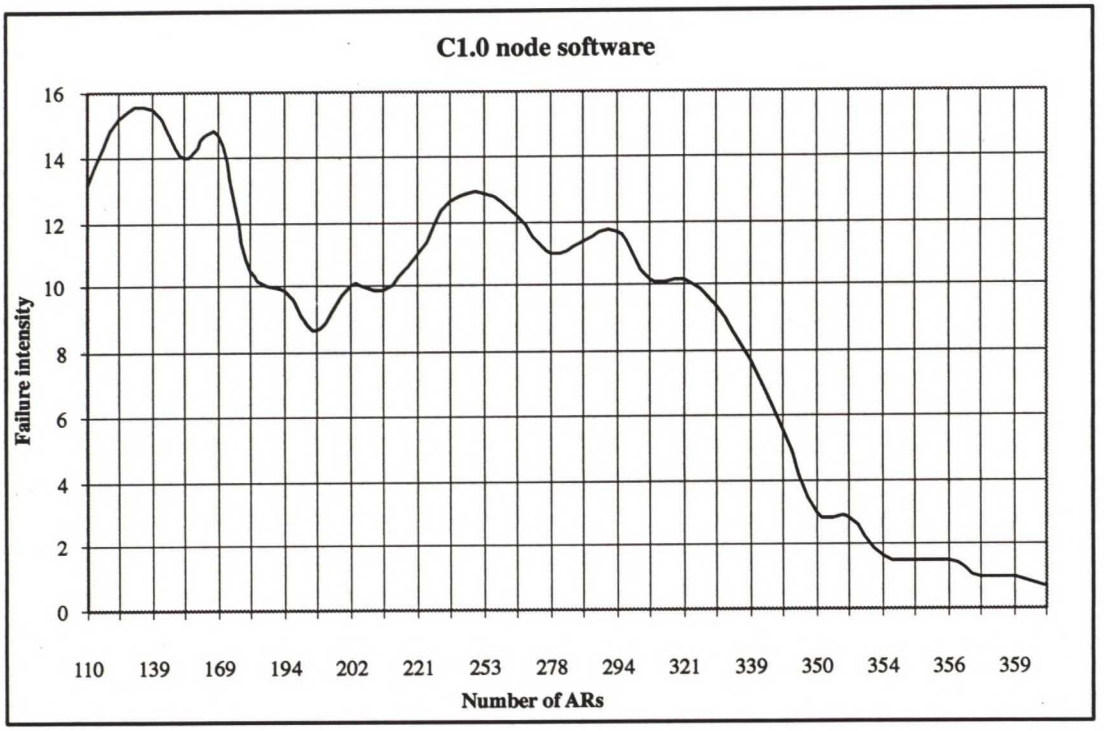


Figure 29. *Synfonet C2.1 & C3.0 node software failure intensity.*

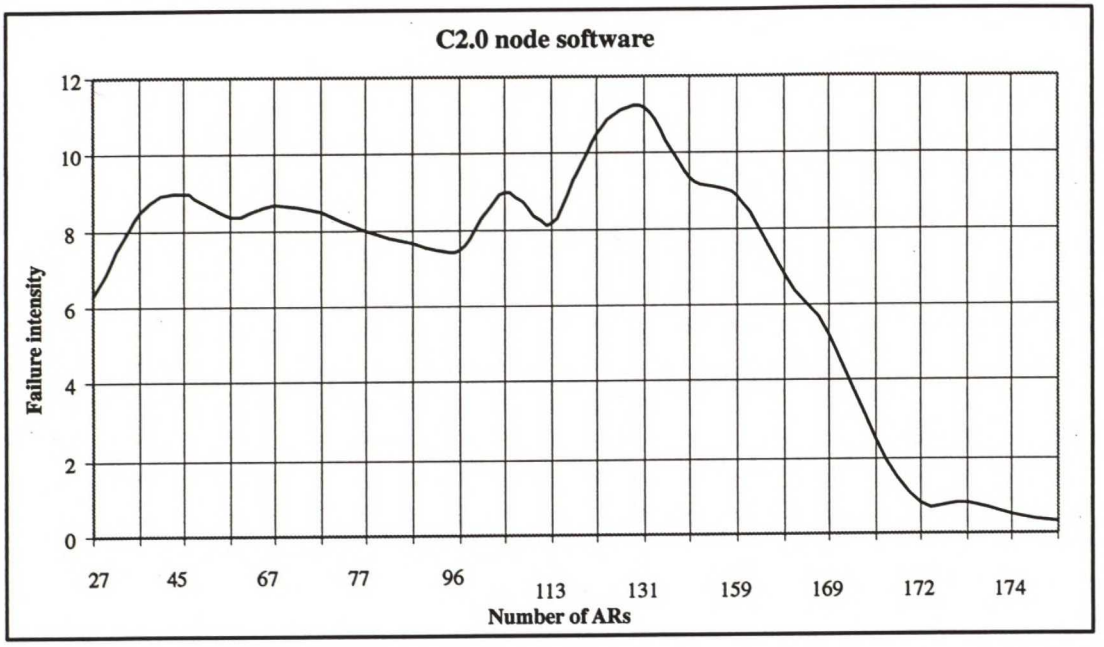


Figure 30. *Synfonet C2.0 node software failure intensity.*

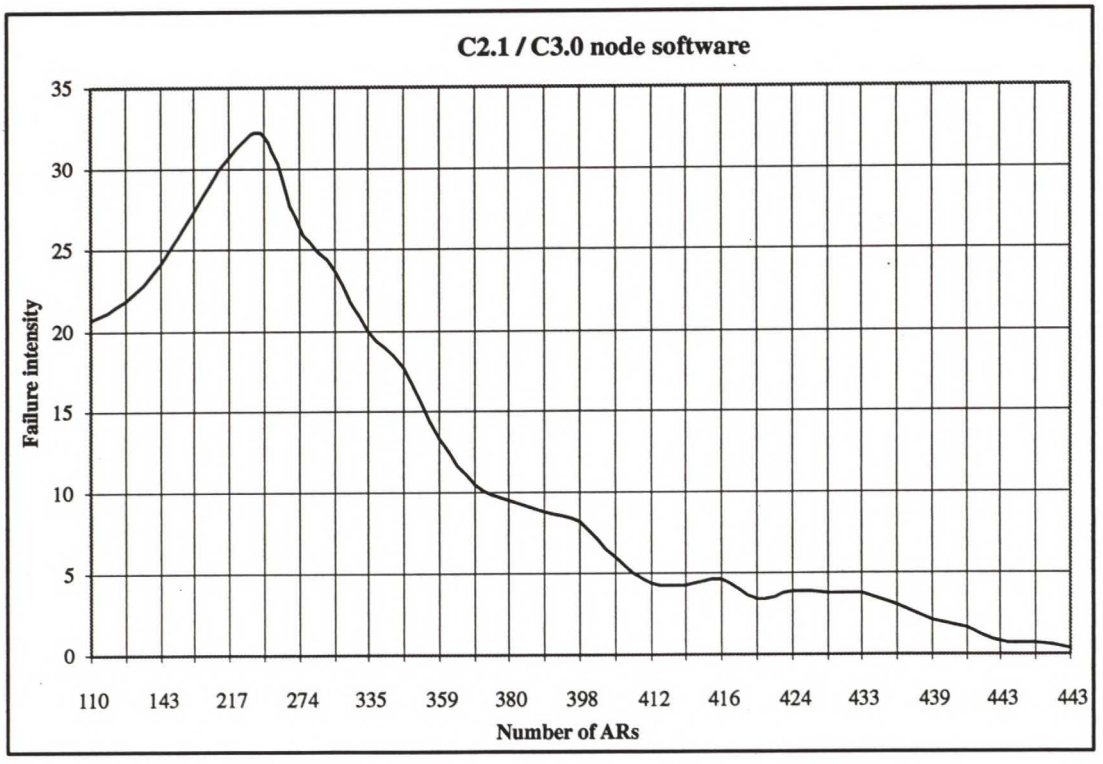


Figure 31. *Synfonet C2.1 & C3.0 node software failure intensity.*

7.6.3 An Example of the Use of Reliability Models

Although the quality of the current failure data is not sufficient for thorough reliability analysis, can an experimental modelling be done. The basic idea is to analyze C2.1 / C3.0 failure data (see Figure 31) and fit both basic model and logarithmic Poisson model to it. This is done by linear regression using Microsoft Excel. A sufficient confidence interval analysis is not done.

The steps of the procedure are like this

1) Select the data point [number of failures x, failure intensity Y] to be the origin. From this point on, the failure intensity is decreasing. The data point is (x_1,y_1). Define $X = x - x_1$. An exact method for defining this point can be created.

2) Fit these models to the data using a certain number of data points (in our case 12)

$$Y = AX + B, \tag{15}$$

$$Y = C \exp(-DX). \tag{16}$$

The estimation can be done by taking natural logarithms on the both sides of (16). Then the equation becomes

$$\ln(Y) = \ln(C) - DX. \tag{17}$$

In equations (15)–(17), X is cumulative number of failures, and Y is failure intensity. In calculating Y, X is used. So X and Y are not totally independent. This is one source of estimation errors. 12 first data points have been used for estimation. In our case the point [217, 33] has been selected to be the origin. Table 6 represents the estimated parameters. From parameters we can get the estimates of the intial failure intensity λ_0, the total number of failures ν_0, and the failure intensity decay parameter θ (see equations (2) and (4) in Chapter 6).

Table 6. *Parameter estimates for two models*

Model	*Parameter*	*Estimate*	*CI 95%*	R^2
Basic	***A***	–0.17	[–0.27; –0.08]	***0.66***
	B	39.3	[26.4; 52.2]	
log Poisson	***ln(C)***	4.04	[3.15; 4.93]	***0.63***
	D	11.1 E–3	[4.7 E–3; 17.6 E–3]	

The results of the estimation can be seen in Figure 32.

The estimate of λ_0 is thus 39.3 for the basic model and 56.8 (=exp(4.04)) for the logarithmic Poisson model. If equation (15) is written in same format as (2), we can get the estimate of ν_0, that is 231. If we remember the scaling $X=x-x_1$, the estimate corresponds 231+217=448 total failures in the real life. This is quite near the observed amount of total failures 443. So we could speculate that there are still five more failures to be found! The estimate of the failure decay parameter θ is 11.1 E–3.

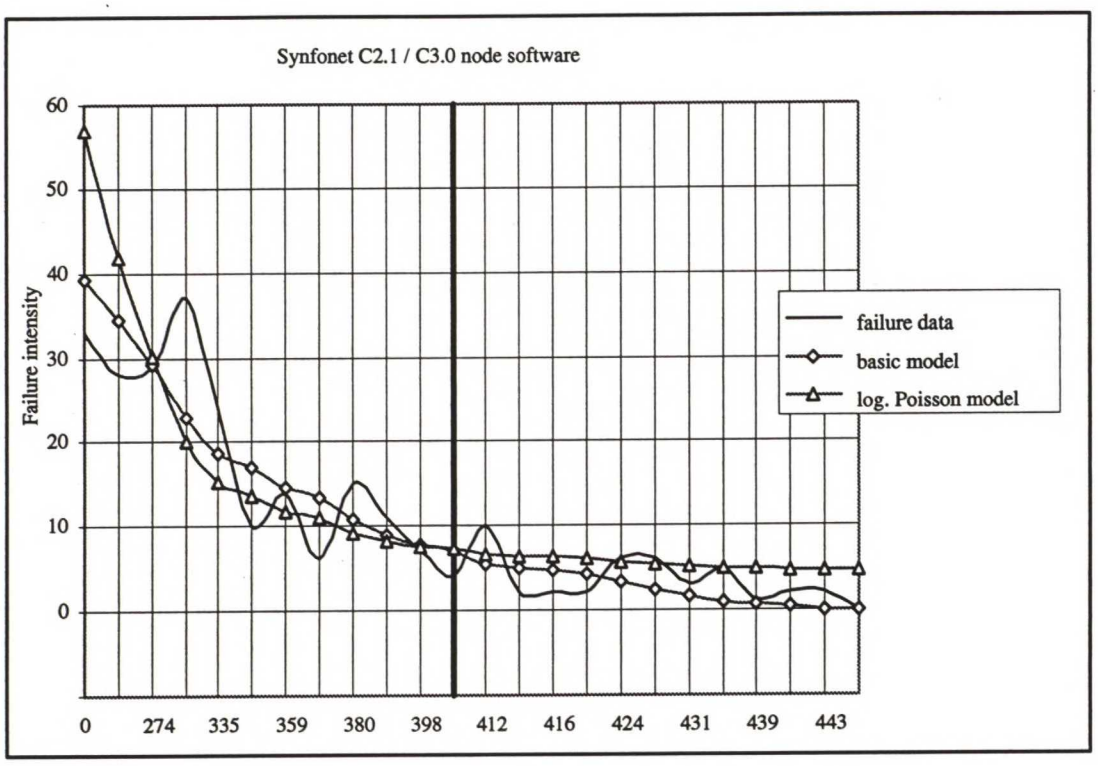


Figure 32. *Basic model and logarithmic Poisson model fitted to Synfonet C2.1/C3.0 node software failure data.*

What can be said about this example? It simulates the case we have observed twelve data points and we believe that the failure intensity is a decreasing function of the cumulative number of failures. Then we fit two models to the data and try to estimate the total number of faults in the code. We also could estimate the time when there will be less than N failures in the code at some level of risk.

The R^2 statistics are not very good for the models (0.66 and 0.63). R^2 measures the proportion of the variation in Y which is "explained" by the regression equation. It is often informally used as a goodness–of–fit statistic, but we must remember that all statistical results follow from the initial assumption that the model is correct, and R^2 is sensitive to the number of independent variables included in the regression model.

The different natures of the models can be seen in Figure 32. In the basic model, the decrement in the failure intensity function remains constant. In the logarithmic Poisson model, the failure intensity drop is logarithmic (first very fast and then slower). The 12 data points used in the estimation are on the left side of the thicker black line.

7.6.4 Impact of the Code Size on the Number of Failures

Implementing new functionalities nearly always increases the size of the code. Figure 33 represents the size of the code in different Synfonet node software releases. The size of the code is usually measured in lines of code.

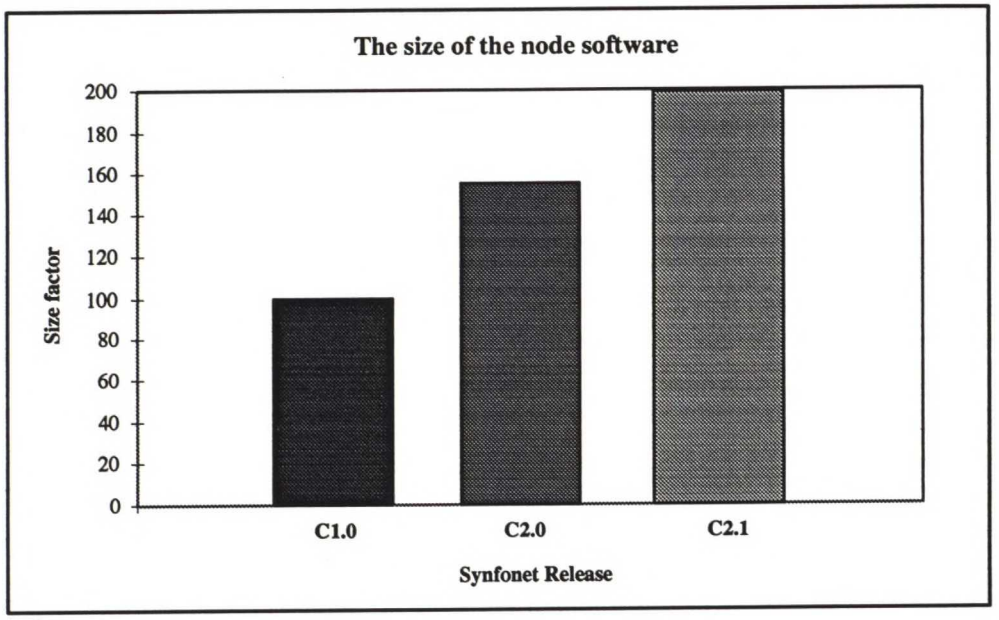


Figure 33. *The size of the node software in different releases (lines of code), C1.0 scaled to 100.*

If we compare Synfonet C1.0 and C2.1, it can be seen that the size of the code has almost doubled. The size of the code also has a great effect on the testing effort needed.

Using the information received from the software department, it can be seen that the change from C2.0 to C2.1 was not a minor one. In this case, the first commercial releases of the Synfonet node softwares C2.0 and C2.1 have been compared. These counts are approximate:

Added lines of code	400 000
Removed lines of code	35 000
Modified lines of code	115 000

The number of faults in the code is a function of the lines of code. Assuming that the number of faults/line of code is constant, the total number of failures can be modelled as a Poisson process, i.e. the number of failures (k) in a code of size T is Poisson distributed with parameter $\lambda \cdot T$. The point estimate for λ is

$$\lambda = k / T. \qquad (18)$$

The 95 per cent confidence intervals for λ can be calculated using formulas [13]:

$$\lambda_{lower} = \chi^2_{0.025}(2k) / 2T, \text{ and } \lambda_{upper} = \chi^2_{0.975}(2k+2) / 2T. \quad (19)$$

χ^2 (n) is a point of χ^2 distribution with n degrees of freedom.

In our case we assume that each node software version has been tested thoroughly, so nearly all possible failures have been found. The results are in Table 7.

Table 7. *Poisson parameters and their 95 % confidence intervals.*

Release	*Node SW ARs*	*Code size factor*	*Poisson parameter*	*CI 95 %*
C1.0	359	100	3.59	[3.24 3.97]
C2.0	183	156	1.17	[1.01 1.35]
C2.1/C3.0	448	199	2.25	[2.05 2.46]

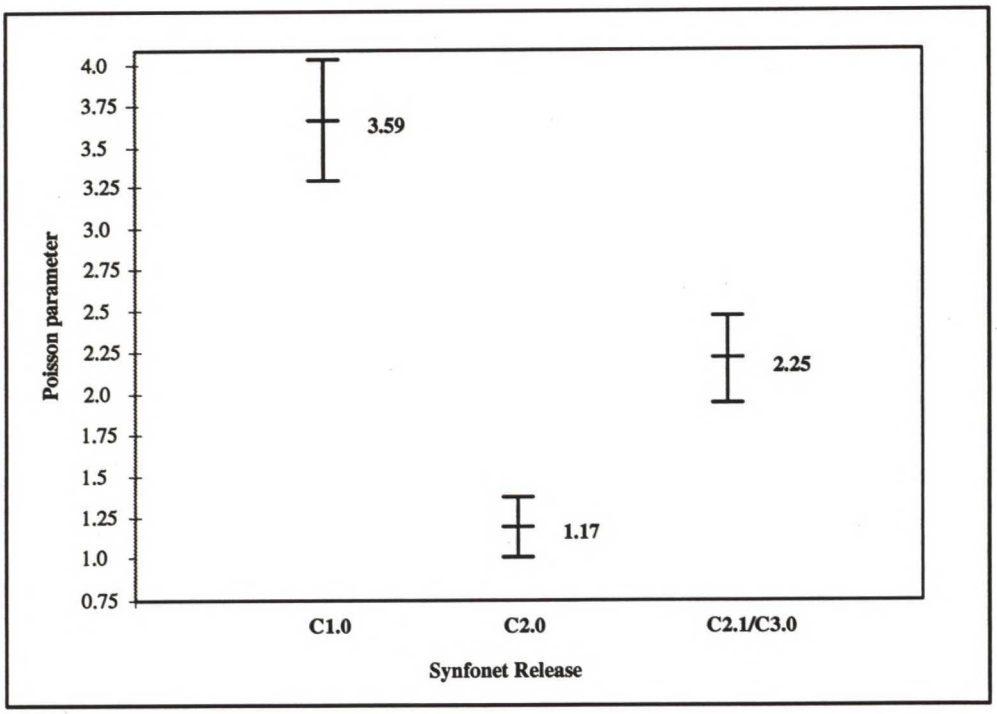


Figure 34. *Poisson distribution parameters with 95 per cent confidence intervals in different node software versions.*

Analyzing Figure 34, we can conclude that there is a significant difference in the fault behaviour of different node software versions. The confidence intervals do not overlap in any case. It is natural that the parameter estimate is biggest in the first version. The estimate is three times smaller in C2.0 and grows again in C2.1/C3.0. These reasons can be thought: there were no major changes between C1.0 and C2.0 and very much functionality was added in C2.1/C3.0, such as hardware unit protection and auxiliary data channels.

7.7 Failures in Synfonet Node Manager

7.7.1 Failure Distributions

The fault reports created for Windows based Synfonet Node Manager are sorted by severity in Figure 35. There is one release of SNM that is capable of managing both C2.1 and C3.0 nodes. That release is SNM C2.1. Major faults represent about 28 per cent of the total faults in each release. Fatal faults are between 16 and 25 per cent. It is noticeable that the number of fatal faults is so low in SNM C2.1. Major and fatal faults together represent about 50 per cent of the total number of failures in each release.

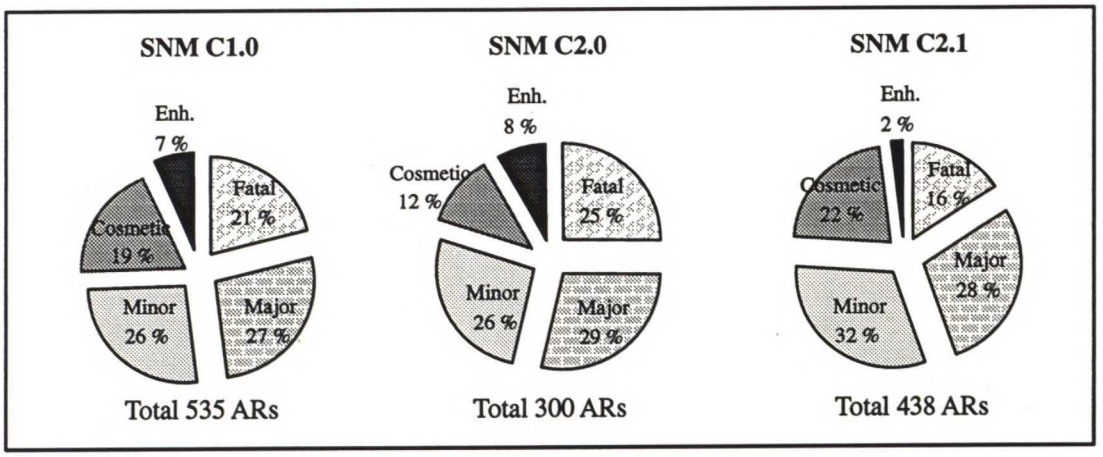


Figure 35. *Synfonet Node Manager failure severity distributions.*

If Figures 19 and 35 are compared, it can be noticed that there are over four times more cosmetic and enhancement ARs in SNM (except enhancements in SNM C2.1). This is natural because cosmetic faults and enhancements are in many case user interface problems. The portions of major faults differ significantly: there are about 1.5 times more major faults in node software releases than in SNM. The behaviour of fatal faults is very similar in both cases. A statistical analysis of the failure distributions is made in Section 7.8.

7.7.2 Cumulative Failures and Failure Frequencies

Cumulative number of failures is represented for three SNM releases in Figures 36–38. It can be seen that only few failure reports are created for during 6–12 first weeks. These ARs are made during software integration. In SNM C1.0, the stabilization in the number of ARs happens after 43 weeks and in SNM C2.0 it happens already after 19 weeks. In SNM C2.1 it happens after 35 weeks. There are differences in the shapes of the curves but the basic format is the same.

SNM version C1.0 was programmed in Finland and SNM versions C2.0 and C2.1 were programmed in Cambridge Product Development (UK). SNM version C1.0 (and C1.0 nodes, of course) did not have the Q3 stack (management interface). An another fault reporting system was used in SNM C2.0 integration test. Fault reports from the another system were copied to Action Request System. Beginning from SNM C2.1, only ARS is used.

Because SNM is always tested together with the nodes, it is very important to know what component caused the failure situation. Some additional software tools are used in system/integration testing to find out whether the problem is in the node or in SNM. One tool can communicate directly with the node software layers THD and NFL and another tool communicates with the node via ASW.

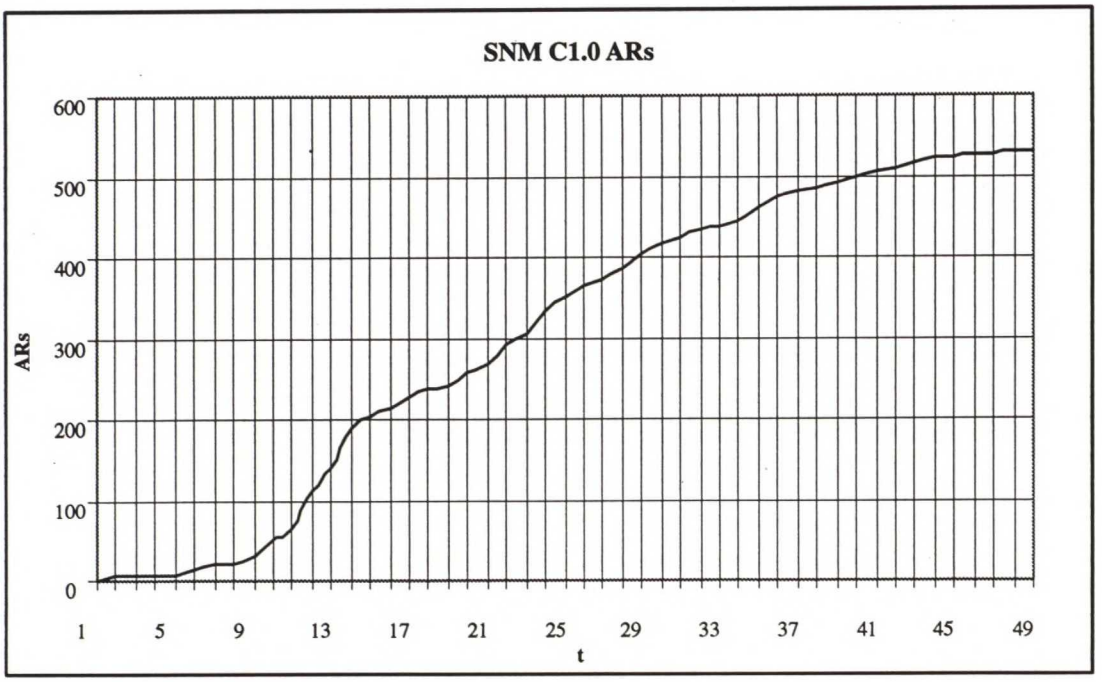


Figure 36. Cumulative number of failures (ARs) in SNM C1.0.

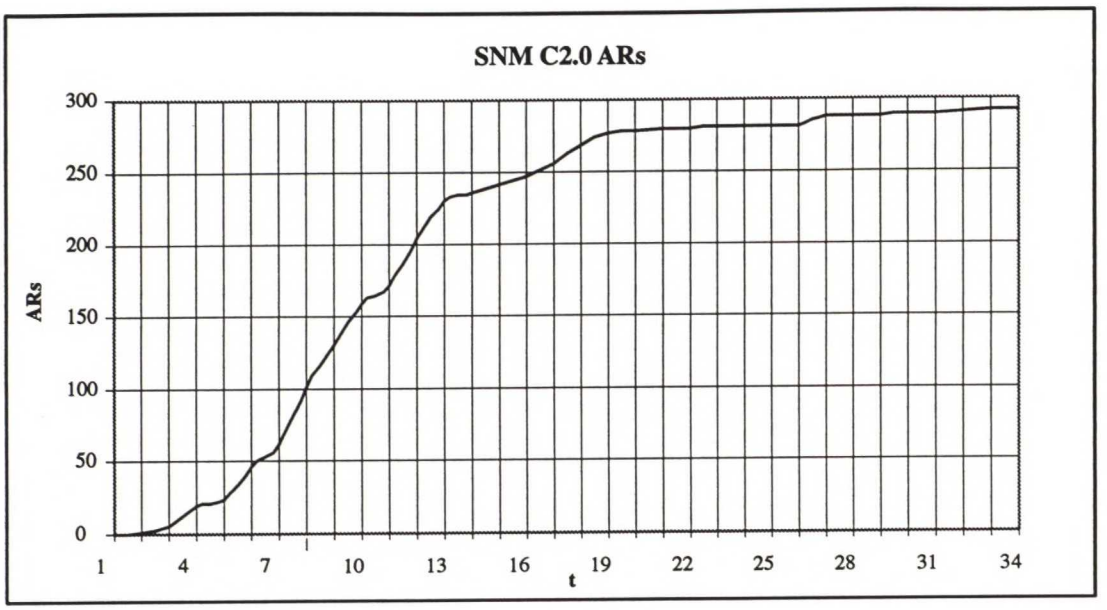


Figure 37. *Cumulative number of failures (ARs) in SNM C2.0.*

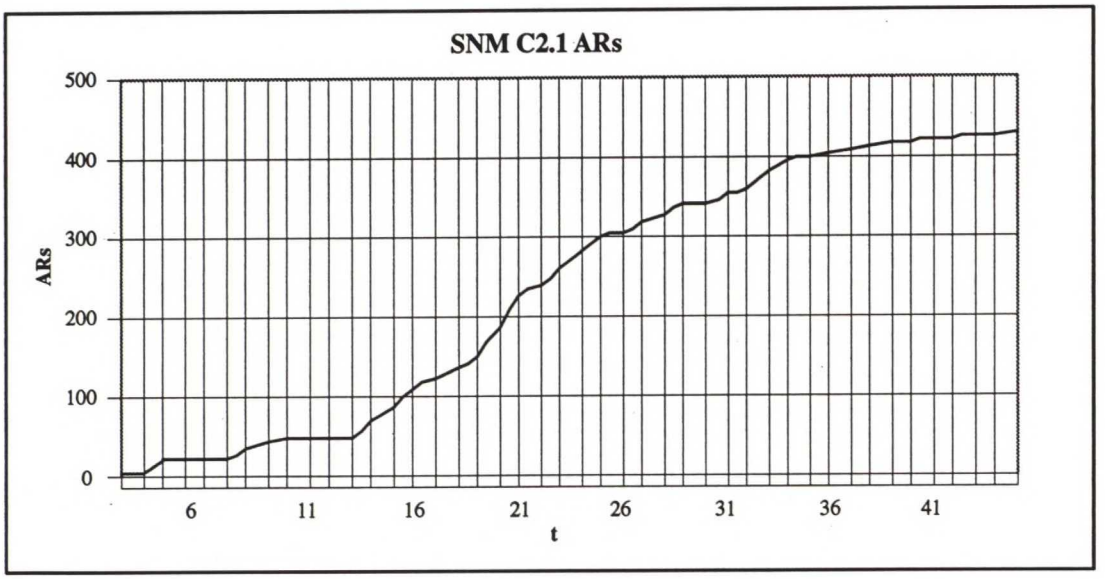


Figure 38. *Cumulative number of failures (ARs) in SNM C2.1.*

Figures 39–41 represent the failure intensities in different SNM releases. Like in the node software (Figures 26–28), the failure intensity curve of the release C2.1 is the most symmetric. The other fault curves behave much like the node software curves. SNM versions C1.0 and C2.0 differ from SNM C2.1 in one way: SNM C1.0 and C2.0 reach their maximum intensity after 11–13 weeks, but the maximum intensity in SNM C2.1 is reached after 23 weeks. If compared to node software, SNM C2.1 reaches its maximum intensity almost simultaneously with the node software, but SNM versions C1.0 and C2.0 reach it over ten weeks before.

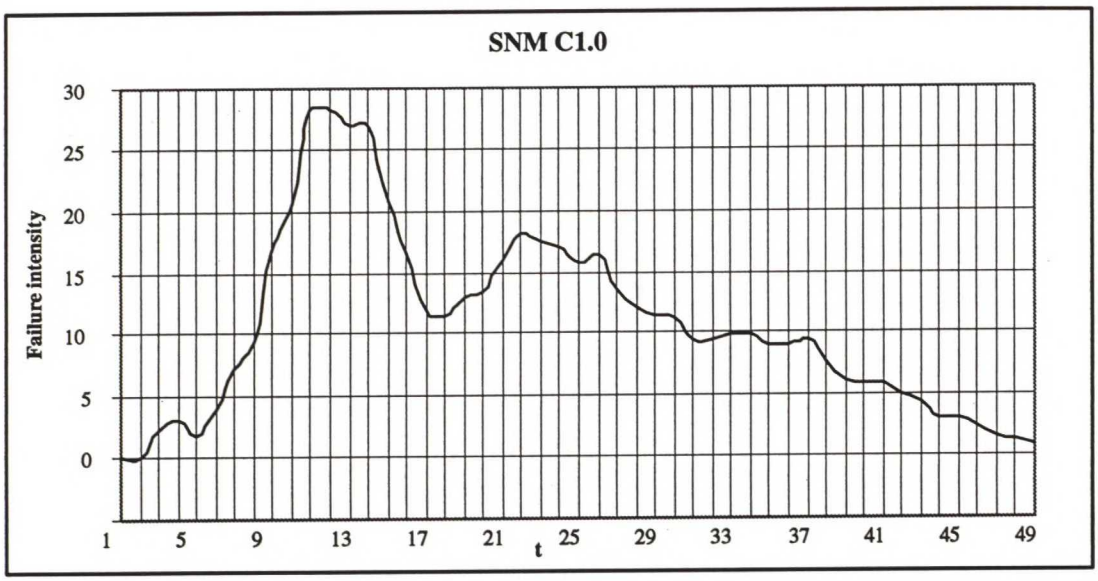


Figure 39. *SNM C1.0 weekly failure intensity.*

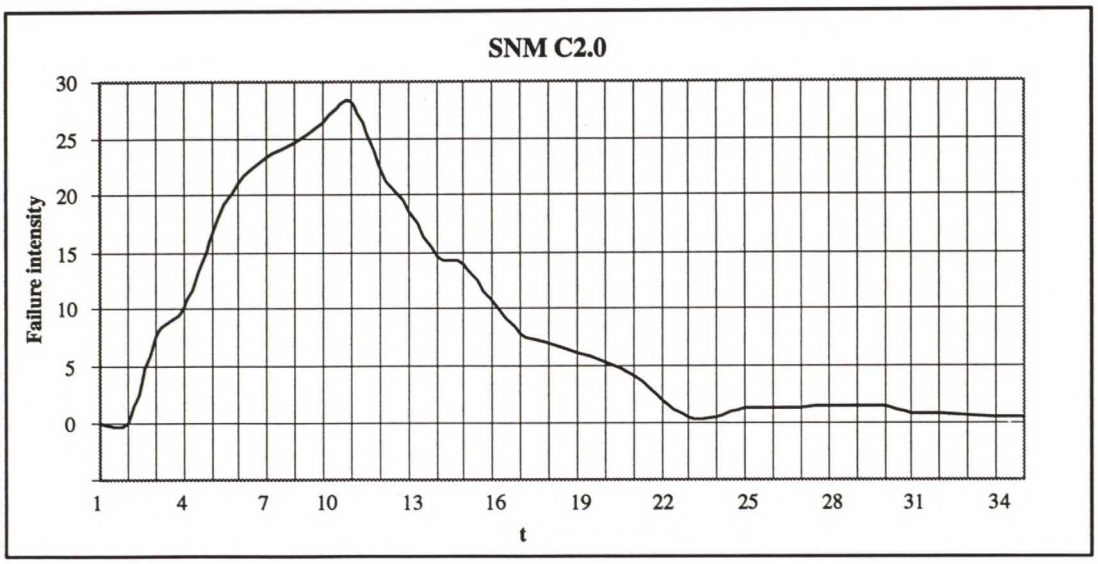


Figure 40. *SNM C2.0 weekly failure intensity.*

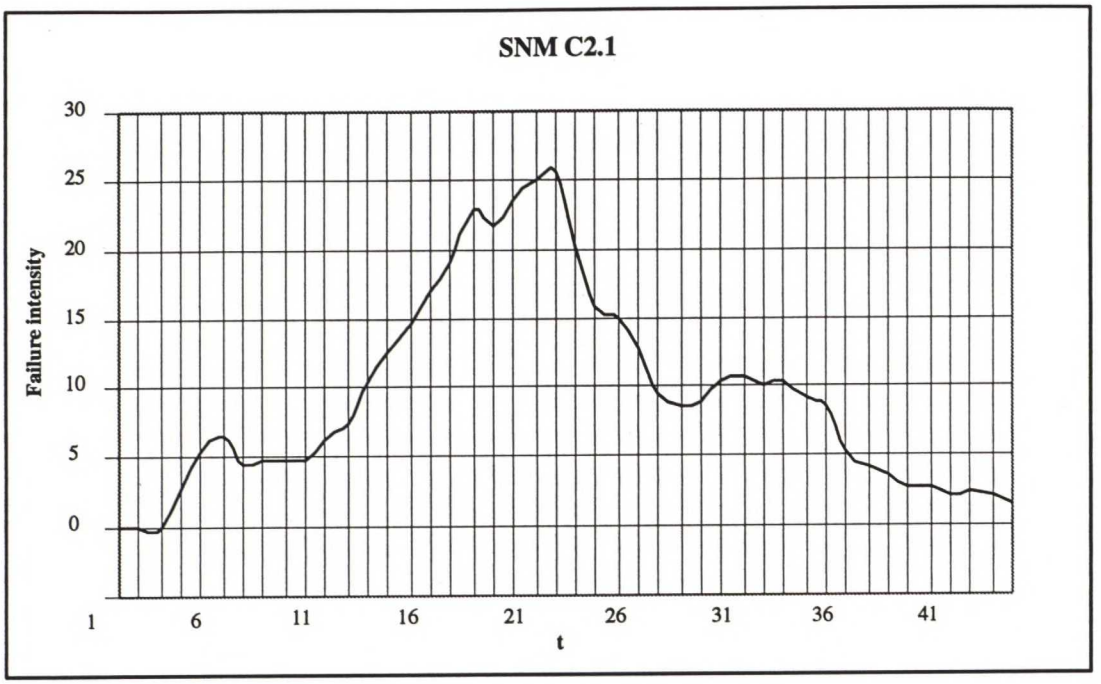


Figure 41. *SNM C2.1 weekly failure intensity.*

7.7.3 Impact of the Windows Platform

The platform is very important in developing PC software. Microsoft Windows has been selected to be the platform for node management applications. The PC based network management application NMS/10 also runs on Windows platform.

SNM C1.0 was developed for Microsoft Windows 3.1. SNM C2.0 and SNM C2.1 are able to run on both Windows 3.1 and 3.11. The latest version of SNM C2.1 can also be used in Windows NT. The platform for next Synfonet releases is Windows 95 and Windows NT (3.51/4.0). The change in Windows platform is a challenge for programmers and also for system test department. The programmers must learn the new programming/designing features of the new platform. The test engineers must have a good knowledge of the new platform to be able to test the product in that environment. When updating SNM from C2.1 to C2.20, the operating system of the PC must first be updated to Windows 95. Only after that can the SNM update be done.

The Windows NT version of SNM C2.1 made it possible to manage several nodes simultaneously using one PC. The NT version made it also possible to use SNM in a UNIX workstation. This made the co–operation of NMS/100 and SNM C2.1 possible. In the UNIX workstation, SNM runs on WinDD server (Windows Distributed Desktop, trademark of Tektronix).

The stability of the platform has a great effect on the stability of the application. The platform also gives the performance limits of the application. If a fatal failure occurs in a Windows program, the program is closed and Windows gives an error message of a General Protection Fault, Application Error or Illegal Instruction. If a GPF occurs in Windows 3.x, the Windows is usually corrupted and all other programs running should also be closed. The PC and Windows must be restarted. The situation is better in Windows 95 / NT. The crash of one application does not disturb the other applications, and the user does not have to restart the PC. This saves very much valuable testing time. If the tested application crashes, say, 15 times a day and the PC restarting takes three minutes, 45 minutes testing time per day is lost.

In Windows 95 and NT the tracing of the faults in the code is also easier. It can be said that a good platform supports the application and does not crash the application in all possible fault situations.

7.8 Statistical Analysis of Failure Distributions

In statistical testing, two hypotheses are compared: null hypothesis H_0 and an alternative hypothesis H_1. A test statistic is calculated using the data. The P–value of the test statistic tells how probable the value of the test statistic is when H_0 is true. The P–value is compared to the risk level α. Usually, the risk level $\alpha = 0.05$ is used. If the P–value is lower than the selected risk level, H_0 is rejected. [14, p.23].

In this analysis, the chi–squared (χ^2) test is used. The test is done analyzing the contingency tables [14, p.27]. The hypotheses are defined this way:

H_0: The data is homogeneous i.e. there is no significant difference in the data distributions.

H_1: The data is not homogeneous. There is a significant difference in the data distributions.

The actual frequences (fault distributions) are put in a contingency table. The test statistic is calculated using the formula [14, p.27]:

$$\chi_0^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} (O_{ij}-E_{ij})^2 / E_{ij}, \quad df = (r-1)(c-1). \tag{20}$$

In (20), r is the number of rows in the contingency table, c is the number of columns, O_{ij} is the actual frequence in cell (i,j) and $E_{ij} = (\sum_i O_{ij})(\sum_j O_{ij})/(\sum_i \sum_j O_{ij})$ is the expected frequency in cell (i,j). The P–value is $P(\chi^2 \geq \chi_0^2)$.

Following tests are done. Different node software versions are compared to each other and node software is compared to SNM. The results are in Table 8. The P–values have been calculated using Microsoft Excel 'Chitest' function.

1. All node software versions (C1.0, C2.0 and C2.1/C3.0) are analyzed. It is verified whether the distribution of failures in groups THD, NFL and ASW differ significantly (contingency table size 3x3).

2. SNM C2.1 and C2.1/C3.0 node software are compared. It is verified whether the distribution of failures in all severity classes differ significantly (table size 2x5).

3. SNM C2.1 and C2.1/C3.0 node software are compared. The comparison of failure distributions is done in severity classes Fatal, Major and Minor (table size 2x3).

4. C1.0 and C2.1/C3.0 node software is compared. The comparison of failure distributions is done in severity classes Fatal, Major and Minor (table size 2x3).

5. C2.0 and C2.1/C3.0 node software is compared. The comparison of failure distributions is done in severity classes Fatal, Major and Minor (table size 2x3).

6. All SNMs are compared. Failures are divided in two groups: A) Fatal and Major, and B) Minor, Cosmetic and Enhancement. (table size 3x2).

***Table 8.** Results of the analysis of the contingency tables.*

Test case	*df*	*P–value*	*Reject H_0*
1.	4	9.72 E–3	Yes
2.	4	1.32 E–13	Yes
3.	2	0.051	No
4.	2	0.699	No
5.	2	170 E–6	Yes
6.	2	0.056	No

We conclude that there is significant difference in the fault distributions of different node software versions when the groups THD, NFL and ASW are concerned. It means that the percentages of faults in THD, NFL and ASW are not the same for three different versions. This is understandable when Figure 20 is analyzed. The percentages of NFL and THD faults differ a lot in different releases.

Test cases 2 and 3 are interesting. SNM and node software fault distributions are compared in release C2.1/C3.0. If all fault groups are analyzed, the difference is significant, but if only fatal, major and minor (i.e. the most important faults) are examined, the difference is not significant (although 0.051 is very near of the rejection limit 0.05). The major differences are thus in the number of cosmetic and enhancement failures.

In test cases 4 and 5, C2.1/C3.0 node software is first compared to C1.0 and then to C2.0. The result is that C1.0 and C2.1/C3.0 severity distributions (fatal, major, minor) behave in the same way, i.e. they are homogeneous. Synfonet C2.0 clearly differs from two other releases. One possible reason is that there was not very much new functionality added in release C2.0. And the total number of failures is very small in C2.0 node software compared to C1.0 and C2.1/C3.0.

In test case 6, SNM failures are put in two categories: A) Fatal and Major, and B) Minor, Cosmetic and Enhancement. The distribution of failures in these two groups is compared in all SNM releases. The result is, that there is no significant difference (when groups A and B are concerned) between different SNM versions.

7.9 Collecting Failure Data in Next Releases

More effective failure data analysis is needed in next releases. I suggest this kind of weekly follow-up for failure data. The basic idea is making weekly software reliability reports that are made for both Synfonet node software and Synfonet Node Manager. The data collection point could be for example every Friday at 14 o'clock.

First data to be reported is the resources used for testing both softwares, i.e. how many persons in the system/integration test group have been testing the product. Every week data is written down for both softwares. Usable program is for example Microsoft Excel. The following data is stored:

- The number of new failure reports (ARs) created during one week (from Friday to Friday) sorted by severity, for severities see section 5.3.2
- The cumulative number of ARs for that release sorted by severity
- The total number of ARs belonging to the groups 'New', 'Active' or 'Evaluated' (i.e. the existing faults)
- The total number of ARs belonging to the groups 'Fixed', 'Verified' or 'Closed' (i.e. the removed faults)

In integration test or system verification stage a simplified test specification is created. It is called Verification test or System test acceptance test. This test specification is to help in monitoring software reliability. There are two test levels that can be passed or failed in the verification tests:

- Basic test (i.e. it is possible to make this software functionality working in basic conditions, e.g. make the signal go through in a node)
- Stress test (i.e. there are no known ways to create a failure in this functionality; software is thus very robust and error tolerant)

This information is included in each week report:

- Completed cases in verification test specification
- Completed cases in system test specification
- 5–10 most important existing failures for
  - Synfonet node software
  - Synfonet Node Manager

In making decisions, which failures are the most important, the expert information of the system test group is used. The system test group creates this list together. Also the software versions used in testing is put in the report.

7.10 System Test Plan for Synfonet C2.20 / C3.20

Some basic nodes are tested in the next Synfonet release. For these nodes e.g. complete installation tests, configuration tests and card replacement tests are carried out. Table 9 represents the basic node types.

Table 9. *Basic node types used for Synfonet C2.20 / C3.20 testing.*

NODE TYPE ➧	REGs	TM1	TM1E	TM1	TM4	TM4	TM4	ADM	DXCN	DXCN	DXCN
Rack Width	All	19	17	12	19	17	12	19	19	17	12
HW Level	All	1	2	1	2	1	2	1	2	1	2

Several functionalities have to be tested in the next Synfonet release. The functional tests are divided in the following main groups.

1) Backups
2) Synchronization
3) Cross Connect Functions
   - Path protection (SNC)
   - Protection switch speed
   - VC–4 <–> VC–3 <–> VC–2 <–> VC–12
4) HW Protection
   - CU HW protection
   - SSW HW protection
   - TA (2MTA, A2MTA, 34MTA) HW protection
5) Data Channels (incl. SU–A)
6) Events & Alarms
7) A2M Tests
8) 34M Interface Tests
9) 34M & 2M Block Tests
   i.e. Terminal bus block tests
10) STM1E <–> 140M
11) Link up/downgrade (installing new cards)
   - STM4 <–> STM1
   - STM1E <–> 140M
   - (A)2M(TA) <–> 34M
   - 34M <–> STM1 / STM4

   Test done in many node types
12) HW Compatibility
13) FTAM software download protocol
14) C2.0, C2.1 and C2.20 Co–operation
15) System Release Upgrade –> x.20
   - Measurement of transmission errors
   - Compatibility, e.g. CC tables
16) SNM Functionality

17) Q3 Routing
18) Network Test
    - General network configuration
    - Co–operation with PDH Equipment
19) Network Longtime
20) Network Management Test
    - NMS/10
    - NMS/100
    - Use of routers

The system test is started after the system verification test is passed. Before the start of the formal tests, very useful random testing is done in different node types.

8 CONCLUSIONS

In the master's thesis software reliability theory has been applied to SDH software development. The focus has been on system and integration testing. The testing of node and node management software has been introduced in three consecutive commercial releases.

First, a view to SDH and Nokia's Synfonet equipment was given. Different software testing strategies and types were introduced and placed in Nokia Telecommunications product process milestone model. An introduction to defect management (Action Request System) was given.

Fault report writing instructions were completed in chapter 5. Using these instructions, it is possible to create efficient and informative fault reports. The fault report lifecycle was analyzed.

An introduction to software reliability theory was given. Two software reliability models, the basic model and the logarithmic Poisson model, were introduced. Many important concepts of software reliability were introduced.

The software reliability analysis was started with failure definitions for SDH equipment. Failure types were defined for node and node management software. So the negative specification of the system was written. Failure database was efficiently used and analyzed: Synfonet releases C1.0, C2.0 and C2.1/C3.0 were compared to each other. It was noticed that the failure intensity never has its maximum value in the beginning of the fault reporting. Some statistical tests were done in comparing the releases and the impact of the code size on the number of failures was analyzed. The current failure data is not sufficient for applying software reliability models. The situation will be corrected in future releases by improving the quality of the data. However, one experimental modelling was done with moderate results.

A proposal for weekly software reliability follow–up system was created. This system uses efficiently the information of Action Request Database and the expert information of the system test team. Every week a follow–up report is created including the list of the most severe faults at the moment and an analysis of the fault reports. This will help the project management to make decisions on the state of the software. This will also help in priorizing the faults. This information will make it possible to apply the software reliability models in the future. Finally, the system test plan for the next Synfonet release was created.

REFERENCES

[1] ITU–T Recommendation G.707 (Draft): Network Node Interface For the Synchronous Digital Hierarchy (SDH). 1995.

[2] Nokia Telecommunications. Synfonet – Node Equipment for Synchronous Digital Hierarchy Networks, Release C2.0, Product Description. 1995.

[3] Koivusalo, E. Synfonet STM 1/4 C2.1 SW Overview. Nokia Telecommunications. 1996. For internal use only.

[4] Nokia Telecommunications. AP Product Process Manual. 1995.

[5] Kaitajärvi, R. Synfonet Access Node System Test Process Decription. Nokia Telecommunications. 1996. For internal use only.

[6] Lew, K. Installing Quality into Software Development. GDB (The Genome Database) 1994. In WWW "http://gdbdoc.gdb.org/dev/qa/instl_qualTOC.html".

[7] Smith, N. Software Process Development: Observation Reporting and ARS, SDH008. Nokia Telecommunications. 1996. For internal use only.

[8] Bastani, F. Foreword: Software Reliability. IEEE Transactions on Software Engineering. November 1993, pp. 1013–1014.

[9] Neufelder, A. M. System Software Reliability Participant Guide. USA 1995. SoftRel (Reliability Analysis Center Course Material).

[10] Musa, J. D., Iannino, A. and Okumoto, K. Software Reliability Measurement, Prediction, Application, Professional Edition. New York 1990. McGraw Hill Publishing Company, 291 p.

[11] Snyder, D. L. Random Point Processes. New York 1975. John Wiley & Sons, 482 p.

[12] Excel Partnership, Inc. Training courses relating to international standards ISO 9000, QS–9000 and ISO 14000. 1996. In WWW "http://www.tregistry.com/ttr/excel/s013fmea.htm".

[13] Procaccia, H. Guidebook on the Effective Use of Safety and Reliability Data. European Safety Reliability & Data Association (ESReDA). Paris 1995. Société Française d'Etudes et Réalisations, 399 p.

[14] Laininen, P. Formulas and Tables of the Applied Probability Calculus. Helsinki University of Technology (Mat–2.090 Course Material). Espoo 1992.

[15] Newcombe, P. Synchronous Transmission Systems. London 1990. STC Telecommunications, 109 p.

[16] Sexton, M. & Reid, A. Transmission Networking: SONET and the Synchronous Digital Hierarchy. Norwood, MA, USA 1992. Artech House, 360 p.

[17] Chow, M.–C. Understanding SONET / SDH Standards and Applications. New Jersey 1995. Andan Publisher.

[18] Nokia Telecommunications, Software Testing, SWPD–099. Designer's Handbook, Part VIII, Software Development Guide. 1993. For internal use only.

[19] Chehab, R. Testing of Q3 Message Routing in SDH Networks. Helsinki University of Technology. Espoo 1995.

APPENDIX 1: AN EXAMPLE OF FINDING A SOFTWARE FAILURE AND WRITING A FAULT REPORT

Number : 14–02555
Date : 01/23/96 15:39:58
Submitter : wiljakka
Title : Changing 2MTA to some other card does not delete the Unit Protection group.
Category : SYNFONET — Node SW — NFL — UPH
Details :

01/23/96 15:39:58 *wiljakka*

You have 2 2MTA's installed e.g. in slots 6 and 7 in 17 slot DXC node and 2MTA protection group is created. After that if you change the Expected Type of slot 6 to 2M, the UP group is not deleted. Minor 'Loss of Protection, missing TA' alarm is left on and you can check the actual UPM situation with EtSW. This also leads to incorrect info in SNM (e.g. 'Protecting U–27392 (null) Failed' in Unit prot. window) and maybe GPFs.

01/23/96 16:02:35 *pschultz*

Forwarded to jarre to check whether this is a problem with UPH or UPM.

01/26/96 12:39:35 *jarre*

I tried this in simulator and UPM did not get any delete pgroup messages from UPH. After the change card action (2MTA–>2M) there are a couple of error messages from UPH, perhaps there are some problems between UPH and EQH?
Received a CARD_CONFIG_CHANGE message for slot5:
UPH ERROR same card twice
UPH ERROR Same slot 5 and card twice from EQH

01/28/96 18:54:41 *wood*

Fixed in UPH_2.139.

02/09/96 16:57:04 *wiljakka*

Verified with C2.1_D0.

Status : Closed
Status–History :

New	wiljakka	01/23/96 15:39:58
Active	pschultz	01/23/96 16:02:35
Evaluated	jarre	01/26/96 12:39:35

Fixed	wood	01/28/96 18:54:41
Verified	wiljakka	02/09/96 16:57:04
Closed	wiljakka	02/09/96 16:57:04
Assigned to :	wood	
Owner :	pschultz	
System :	SNM C2.1 X4C01, node sw C2.1_C0	
Severity :	Major	
Reproducible :	Yes	
Priority :	High	
Items affected :	UPH	
Detected :	System Testing	
Found in release :	C2.1	
Fix estimate/hours :	3.5	
Total effort/hours :	8	
Notification :	E–mail + Notifier	

APPENDIX 2: A FAULT REPORT FROM VISE TO ARS DURING MAINTENANCE

Number : 14–02595
Date : 01/29/96 11:03:57
Submitter : laasonen
Title : VISE 171171: SYNFONET NODE MANAGEMENT CONNECTIVITY.
Category : SYNFONET — Node SW — Q3Stack
Details :

01/29/96 11:03:57 *laasonen*

VISE 171171 Author: Kyosti Kuri CCFINHD

Descr.: Customer can not contact nodes locally or remotely by node manager.

Janne Laasonen: That is all the information in VISE. Severity according to VISE.

01/29/96 11:11:21 *hyytia*

This bug exists on deliveried C2.0 node SW and in C2.1/C3.0 node SW.

The bug is related to 32 bit integer which continues to calculate from zero to ... (each time in 3 ms) and when it gets highest value, it stops the Stack.

02/05/96 18:51:31 *tonteri*

Fixed in C2.1/3.0 D0. We verified the operation of the stack in that situation by giving a base value for the tick count so that the clock wrapped around approximately 3 minutes after unit startup. It was tried both on CU and STM units and the operation of the SW was OK. A release will be made if necessary also for C2.0 to fix this problem.

02/15/96 06:51:07 *laasonen*

Closed. VISE 171171 corrected 15.2.96

Status : Closed
Status–History :

New	laasonen	01/29/96 11:03:57
Active	laasonen	01/29/96 11:03:57
Fixed	tonteri	02/05/96 18:51:31
Closed	laasonen	02/15/96 06:51:07

Assigned to : raivola
Owner : laasonen
System : C2.0.1
Severity : Major
Reproducible : Yes
Detected : After Release / Maintenance
Found in release : C2.0

APPENDIX 3: A NEWS RELEASE ABOUT AN SDH CONTRACT

Nokia to Deliver SDH Solution to Cable London of the UK

(April 17, 1996) Cable London of the UK, and Nokia Telecommunications, have signed a five year contract for the supply of a total SDH solution to meet the needs of business and residential telephony in Cable London's North London franchises. The contract is valued at GBP 15 million.

"Nokia has supplied PDH equipment to Cable London for a number of years, and the new SDH equipment will initially work alongside this to complete our network build, and will then be retrofitted to earlier sites," according to Mr Neil Johnson, Managing Director of Cable London.

"SDH technology is central to Cable London's future strategy. We are a leading edge service provider and Nokia's comprehensive solution will ensure that we remain at the forefront, providing the best possible solutions to our customers. In particular we have been very happy to date with the level of service and support which Nokia provides."

"This agreement strengthens our successful relationship with Cable London," said Mr Kari Suneli, Senior VP of Nokia Telecommunications Access Systems, "It also underlines our competences in the area of SDH technology and highlights our commitment to providing complete solutions for cable operators."

The contract incorporates the purchase of Nokia's SYNFONET SDH solution, as well as ACM2 primary multiplexing equipment, the NMS management system, and a service agreement.

As part of the complete Nokia transmission solution, SYNFONET provides a managed, reliable and cost–effective network, which seamlessly interconnects the telecommunications users with the services of the operator. Nokia currently has 25 SDH customers in 11 countries.

Cable London is one of the leading cable operators in the UK, with a total of five franchises in the north London area. Cable London is jointly owned and financed by two of the world's largest multi–national telecommunications organisations, Comcast and TeleWest.